

Description of STM32H7 HAL drivers

Introduction

STMCube™ is an STMicroelectronics original initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per Series (such as STM32CubeH7 for STM32H7 Series)
 - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio
 - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
 - All embedded software utilities delivered with a full set of examples.

The HAL driver layer provides a simple generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the built-upon layers, such as the middleware layer, to implement their functions without in-depth knowledge how to use the MCU. This structure improves the library code reusability and guarantees easy portability to other devices.

The HAL drivers include a complete set of ready-to-use APIs which simplify the user application implementation. As an example, the communication peripherals contain APIs to initialize and configure the peripheral, to manage data transfers based on polling, to handle interrupts or DMA, and to manage communication errors.

The HAL driver APIs are split into two categories: generic APIs which provide common and generic functions for all the STM32 series and extension APIs which include specific and customized functions for a given family or part number.

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

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

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

This user manual is structured as follows:

- Overview of the HAL drivers
- Detailed description of each peripheral driver: configuration structures, functions, and how to use the given API to build your application.



1 General information

The STM32Cube MCU Package runs on STM32 32-bit microcontrollers based on the Arm® Cortex®-M processors.



2

Acronyms and definitions

Table 1. Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
CAN	Controller area network
CEC	Consumer electronic controller
CMSIS	Cortex Microcontroller Software Interface Standard
COMP	Comparator
CPU	Central Processing Unit
CRC	CRC calculation unit
CRYP	Cryptographic processor
DAC	Digital to analog converter
DLYB	Delay block
DCMI	Digital camera interface
DFSDM	Digital filter sigma delta modulators
DMA	Direct Memory Access
ETH	Ethernet controller
EXTI	External interrupt/event controller
FDCAN	FD Controller Area Network
FLASH	Flash memory
FMC	Flexible memory controller
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB Host Controller Driver
HRTIM	High-resolution timer
HSEM	Hardware semaphore
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controller
LTDC	LCD TFT Display Controller
MDIOS	Management data input/output (MDIO) slave
MDMA	Master direct memory access
MMC	MultiMediaCard
MSP	MCU Specific Package

Acronym	Definition
NAND	NAND Flash memory
NOR	Nor Flash memory
NVIC	Nested Vectored Interrupt Controller
OPAMP	Operational amplifier
PCD	USB Peripheral Controller Driver
PWR	Power controller
QSPI	Quad-SPI Flash memory Interface
RCC	Reset and clock controller
RTC	Real-time clock
SAI	Serial Audio Interface
SD	Secure Digital
SMARTCARD	Smartcard IC
SMBUS	System management bus
SPI	Serial Peripheral interface
SPDIFRX	SPDIF-RX Receiver interface
SRAM	SRAM external memory
SysTick	System tick timer
SWPMI	Single wire protocol master interface
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

3 Overview of HAL drivers

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

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

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

The HAL main features are the following:

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

3.1 HAL and user-application files

3.1.1 HAL driver files

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

Table 2. HAL driver files

File	Description
<code>stm32h7xx_hal_ppp.c</code>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32h7xx_hal_adc.c, stm32h7xx_hal_irda.c, ...</i>
<code>stm32h7xx_hal_ppp.h</code>	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. <i>Example: stm32h7xx_hal_adc.h, stm32h7xx_hal_irda.h, ...</i>
<code>stm32h7xx_hal_ppp_ex.c</code>	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. <i>Example: stm32h7xx_hal_adc_ex.c, stm32h7xx_hal_dma_ex.c, ...</i>

File	Description
<code>stm32h7xx_hal_ppp_ex.h</code>	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs <i>Example: stm32h7xx_hal_adc_ex.h,stm32h7xx_hal_dma_ex.h, ...</i>
<code>stm32h7xx_hal.c</code>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<code>stm32h7xx_hal.h</code>	<code>stm32h7xx_hal.c</code> header file
<code>stm32h7xx_hal_msp_template.c</code>	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<code>stm32h7xx_hal_conf_template.h</code>	Template file allowing to customize the drivers for a given application.
<code>stm32h7xx_hal_def.h</code>	Common HAL resources such as common define statements, enumerations, structures and macros.

3.1.2 User-application files

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

Table 3. User-application files

File	Description
<code>system_stm32h7xx.c</code>	This file contains SystemInit() which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows relocating the vector table in internal SRAM.
<code>startup_stm32h7xx.s</code>	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
<code>stm32h7xx_flash.icf (optional)</code>	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<code>stm32h7xx_hal_msp.c</code>	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<code>stm32h7xx_hal_conf.h</code>	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
<code>stm32h7xx_it.c/h</code>	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in <code>stm32h7xx_hal.c</code>) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
<code>main.c/h</code>	This file contains the main program routine, mainly: <ul style="list-style-type: none">• the call to HAL_Init()• assert_failed() implementation• system clock configuration• peripheral HAL initialization and user application code.

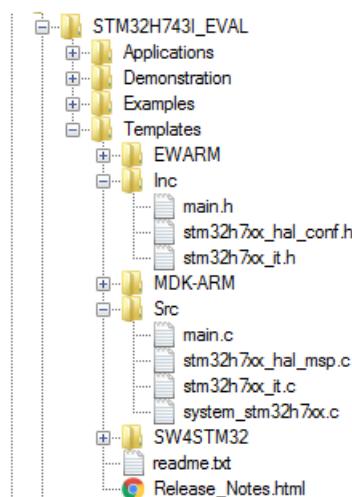
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

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

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

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

Figure 1. Example of project template



3.2 HAL data structures

Each HAL driver can contain the following data structures:

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

3.2.1 Peripheral handle structures

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

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

The peripheral handle is used for the following purposes:

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

An example of peripheral structure is shown below:

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

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

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

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

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

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

3.2.2 Initialization and configuration structure

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

```
typedef struct
{
    uint32_t BaudRate; /*!< This member configures the UART communication baud rate */
    uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received in a frame */
    uint32_t StopBits; /*!< Specifies the number of stop bits transmitted */
    uint32_t Parity; /*!< Specifies the parity mode */
    uint32_t Mode; /*!< Specifies whether the Receive or Transmit mode is enabled or disabled */
}
```

```
        uint32_t HwFlowCtl;          /*!< Specifies whether the hardware flow control mode is enabled or disabled.*/
        uint32_t OverSampling;       /*!< Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8) */
        uint32_t OneBitSampling;     /*!< Specifies whether a single sample or three samples' majority vote is selected */
        uint32_t Prescaler;         /*!< Specifies the prescaler value used to divide the UART clock source */
        uint32_t FIFOMode;          /*!< Specifies if the FIFO mode will be used. This parameter can be a value */
        uint32_t TXFIFOThreshold;    /*!< Specifies the TXFIFO threshold level */
        uint32_t RXFIFOThreshold;    /*!< Specifies the RXFIFO threshold level */
    }UART_InitTypeDef;
```

Note:

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

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

3.2.3 Specific process structures

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

Example:

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

3.3 API classification

The HAL APIs are classified into three categories:

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

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

- **Extension APIs:**

This set of APIs are **family specific APIs** that apply to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

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

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

Table 4. APIs classification

	Generic file	Extension file
Common APIs	X	X

	Generic file	Extension file
Family specific APIs		X

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

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

3.4 Devices supported by HAL drivers

Table 5. List of devices supported by HAL drivers

IP/Module	STM32H743xx	STM32H753xx
stm32h7xx_hal.c	Yes	Yes
stm32h7xx_hal_adc.c	Yes	Yes
stm32h7xx_hal_adc_ex.c	Yes	Yes
stm32h7xx_hal_cec.c	Yes	Yes
stm32h7xx_hal_comp.c	Yes	Yes
stm32h7xx_hal_cortex.c	Yes	Yes
stm32h7xx_hal_crc.c	Yes	Yes
stm32h7xx_hal_crc_ex.c	Yes	Yes
stm32h7xx_hal_cryp.c	No	Yes
stm32h7xx_hal_cryp_ex.c	No	Yes
stm32h7xx_hal_dac.c	Yes	Yes
stm32h7xx_hal_dac_ex.c	Yes	Yes
stm32h7xx_hal_dcmi.c	Yes	Yes
stm32h7xx_hal_dfsdm.c	Yes	Yes
stm32h7xx_hal_dma.c	Yes	Yes
stm32h7xx_hal_dma_ex.c	Yes	Yes
stm32h7x_hal_dma2d.c	Yes	Yes
stm32h7xx_hal_eth.c	Yes	Yes
stm32h7xx_hal_eth_ex.c	Yes	Yes
stm32h7xx_hal_fdcanc.c	Yes	Yes
stm32h7xx_hal_flash.c	Yes	Yes
stm32h7xx_hal_flash_ex.c	Yes	Yes
stm32h7xx_hal_gpio.c	Yes	Yes
stm32h7xx_hal_hash.c	No	Yes
stm32h7xx_hal_hash_ex.c	No	Yes
stm32h7xx_hal_hcd.c	Yes	Yes
stm32h7xx_hal_hrtim.c	Yes	Yes
stm32h7xx_hal_hsem.c	Yes	Yes
stm32h7xx_hal_i2c.c	Yes	Yes
stm32h7xx_hal_i2c_ex.c	Yes	Yes
stm32h7xx_hal_i2s.c	Yes	Yes
stm32h7xx_hal_i2s_ex.c	Yes	Yes
stm32h7xx_hal_irda.c	Yes	Yes

IP/Module	STM32H743xx	STM32H753xx
stm32h7xx_hal_iwdg.c	Yes	Yes
stm32h7xx_hal_jpeg.c	Yes	Yes
stm32h7xx_hal_lptim.c	Yes	Yes
stm32h7xx_hal_ltcd.c	Yes	Yes
stm32h7xx_hal_mdios.c	Yes	Yes
stm32h7xx_hal_mdma.c	Yes	Yes
stm32h7xx_hal_mmc.c	Yes	Yes
stm32h7xx_hal_mmc_ex.c	Yes	Yes
stm32h7xx_hal_nand.c	Yes	Yes
stm32h7xx_hal_nor.c	Yes	Yes
stm32h7xx_hal_opamp.c	Yes	Yes
stm32h7xx_hal_opamp_ex.c	Yes	Yes
stm32h7xx_hal_pcd.c	Yes	Yes
stm32h7xx_hal_pcd_ex.c	Yes	Yes
stm32h7xx_hal_pwr.c	Yes	Yes
stm32h7xx_hal_pwr_ex.c	Yes	Yes
stm32h7xx_hal_qspi.c	Yes	Yes
stm32h7xx_hal_rcc.c	Yes	Yes
stm32h7xx_hal_rcc_ex.c	Yes	Yes
stm32h7xx_hal_rng.c	Yes	Yes
stm32h7xx_hal_RTC.c	Yes	Yes
stm32h7xx_hal_RTC_ex.c	Yes	Yes
stm32h7xx_hal_sai.c	Yes	Yes
stm32h7xx_hal_sai_ex.c	Yes	Yes
stm32h7xx_hal_sd.c	Yes	Yes
stm32h7xx_hal_sd_ex.c	Yes	Yes
stm32h7xx_hal_sdram.c	Yes	Yes
stm32h7xx_hal_smartcard.c	Yes	Yes
stm32h7xx_hal_smartcard_ex.c	Yes	Yes
stm32h7xx_hal_smbus.c	Yes	Yes
stm32h7xx_hal_spdifrx.c	Yes	Yes
stm32h7xx_hal_spi.c	Yes	Yes
stm32h7xx_hal_spi_ex.c	Yes	Yes
stm32h7xx_hal_sram.c	Yes	Yes
stm32h7xx_hal_swpmi.c	Yes	Yes
stm32h7xx_hal_tim.c	Yes	Yes
stm32h7xx_hal_tim_ex.c	Yes	Yes
stm32h7xx_hal_uart.c	Yes	Yes
stm32h7xx_hal_uart_ex.c	Yes	Yes
stm32h7xx_hal_usart.c	Yes	Yes
stm32h7xx_hal_wwdg.c	Yes	Yes
stm32h7xx_ll_delayblock.c	Yes	Yes
stm32h7xx_ll_fmc.c	Yes	Yes
stm32h7xx_ll_sdmmc.c	Yes	Yes
stm32h7xx_ll_usb.c	Yes	Yes

3.5 HAL driver rules

3.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6. HAL API naming rules

	Generic	Family specific	Device specific
File names	<i>stm32h7xx_hal_ppp (c/h)</i>	<i>stm32h7xx_hal_ppp_ex (c/h)</i>	<i>stm32h7xx_hal_ppp_ex (c/h)</i>
Module name	<i>HAL_PPP_MODULE</i>		
Function name	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	NA	NA

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the *stm32h7xx* reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in *stm32h7xxx.h* header file.

Note:

stm32h7xxx.h corresponds to *stm32h743xx.h* and *stm32h753xx.h*.

- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (e.g. ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxConfTypeDef (e.g. ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypeDef (e.g. DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (e.g. HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named PPP_DeInit, e.g. TIM_DeInit.

- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL_PPP_Function_DMA()*.
- The **Feature** prefix should refer to the new feature.
Example: *HAL_ADC_Start()* refers to the injection mode

3.5.2 HAL general naming rules

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

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

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

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below:

Note:

This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7. Macros handling interrupts and specific clock configurations

Macros	Description
<code>_HAL_PPP_ENABLE_IT(_HANDLE_, _INTERRUPT_)</code>	Enables a specific peripheral interrupt
<code>_HAL_PPP_DISABLE_IT(_HANDLE_, _INTERRUPT_)</code>	Disables a specific peripheral interrupt
<code>_HAL_PPP_GET_IT (_HANDLE_, _INTERRUPT_)</code>	Gets a specific peripheral interrupt status
<code>_HAL_PPP_CLEAR_IT (_HANDLE_, _INTERRUPT_)</code>	Clears a specific peripheral interrupt status
<code>_HAL_PPP_GET_FLAG (_HANDLE_, _FLAG_)</code>	Gets a specific peripheral flag status
<code>_HAL_PPP_CLEAR_FLAG (_HANDLE_, _FLAG_)</code>	Clears a specific peripheral flag status
<code>_HAL_PPP_ENABLE(_HANDLE_)</code>	Enables a peripheral
<code>_HAL_PPP_DISABLE(_HANDLE_)</code>	Disables a peripheral
<code>_HAL_PPP_XXXX (_HANDLE_, _PARAM_)</code>	Specific PPP HAL driver macro
<code>_HAL_PPP_GET_IT_SOURCE (_HANDLE_, _INTERRUPT_)</code>	Checks the source of specified interrupt

- NVIC and SYSTICK are two Arm Cortex core features. The APIs related to these features are located in the `stm32h7xx_hal_cortex.c` file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example: `STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)`.

- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

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

- The macros defined below are used:

- Conditional macro:

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

- Pseudo-code macro (multiple instructions macro):

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

3.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32h7xx_it.c
- User callback functions.

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

There are three types of user callbacks functions:

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

Table 8. Callback functions

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

3.6

HAL generic APIs

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

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

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

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

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

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

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

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

Table 9. HAL generic APIs

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

Function group	Common API name	Description
<i>State and Errors group</i>	<i>HAL_ADC_GetState()</i>	This function allows getting in runtime the peripheral and the data flow states.
	<i>HAL_ADC_GetError()</i>	This function allows getting in runtime the error that occurred during IT routine

3.7 HAL extension APIs

3.7.1 HAL extension model overview

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

The extension model consists of an additional file, `stm32h7xx_hal_ppp_ex.c`, that includes all the specific functions and define statements (`stm32h7xx_hal_ppp_ex.h`) for a given part number.

Below an example based on the ADC peripheral:

Table 10. HAL extension APIs

I/O operationg group	Common API Name
<code>HAL_ADCEx_Calibration_Start()</code>	This function is used to start the automatic ADC calibration.

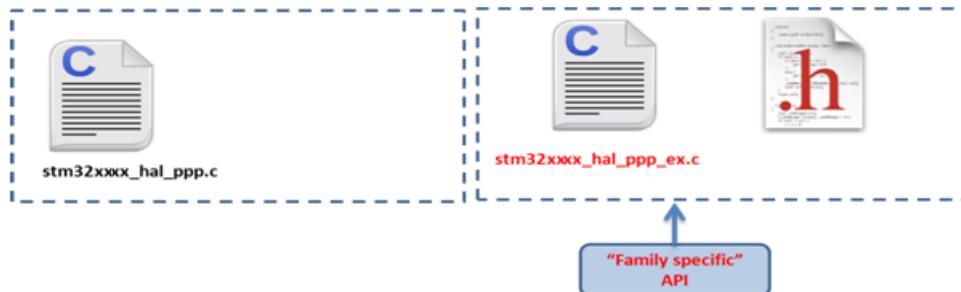
3.7.2 HAL extension model cases

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

Case 1: Adding a family-specific function

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

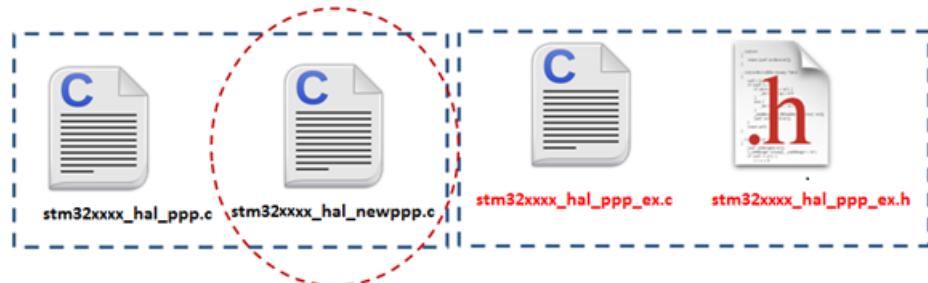
Figure 2. Adding family-specific functions



Case 2: Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32h7xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32h7_hal_conf.h` using the macro:

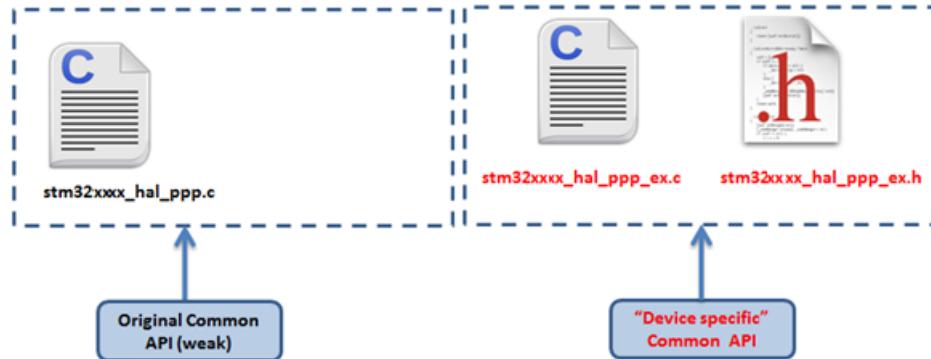
```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 3. Adding new peripherals

Example: stm32h7xx_hal_sai.c/h

Case 3: Updating existing common APIs

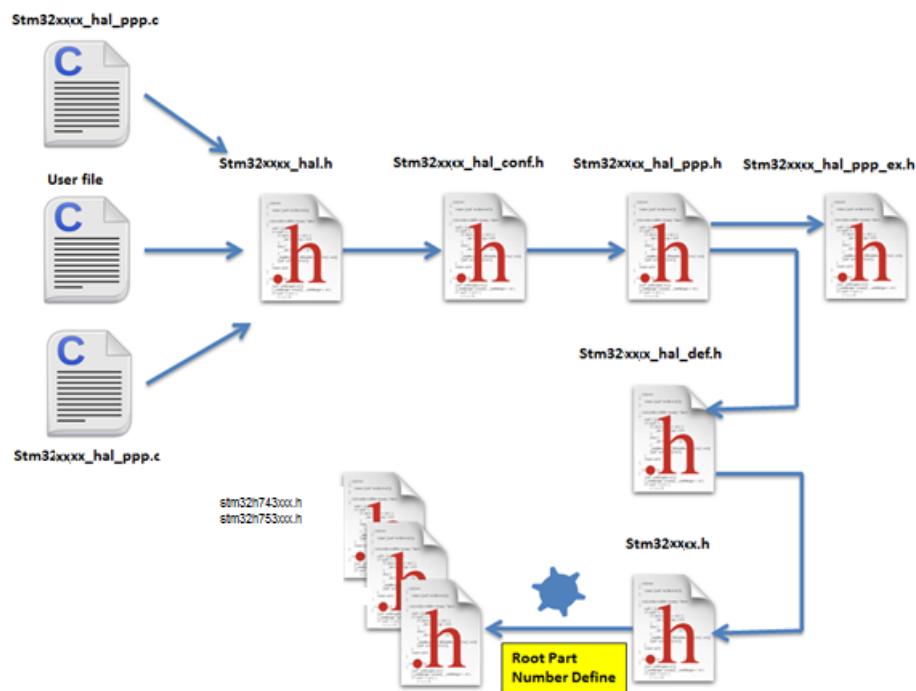
In this case, the routines are defined with the same names in the `stm32h7xx_hal_ppp_ex.c` extension file, while the generic API is defined as `weak`, so that the compiler will overwrite the original routine by the new defined function.

Figure 4. Updating existing APIs

3.8 File inclusion model

The header of the common HAL driver file (`stm32h7xx_hal.h`) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 5. File inclusion model



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

```
/*
 * @file stm32h7xx_hal_conf.h
 * @author MCD Application Team
 * @version VX.Y.Z * @date dd-mm-yyyy
 * @brief This file contains the modules to be used
 */
#define USE_HAL_USART_MODULE
#define USE_HAL_IRDA_MODULE
#define USE_HAL_DMA_MODULE
#define USE_HAL_RCC_MODULE
```

3.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32h7xx_hal_def.h*. The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL Status**

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

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

- **HAL Locked**

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

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

In addition to common resources, the `stm32h7xx_hal_def.h` file calls the `stm32h7xx.h` file in CMSIS library to get the data structures and the address mapping for all peripherals:

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

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

- Macro linking a PPP peripheral to a DMA structure pointer: `__HAL_LINKDMA()`:

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

3.10 HAL configuration

The configuration file, `stm32h7xx_hal_conf.h`, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

Table 11. Define statements used for HAL configuration

Configuration item	Description	Default Value
<code>HSE_VALUE</code>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 Hz
<code>HSE_STARTUP_TIMEOUT</code>	Timeout for HSE start-up, expressed in ms	5000
<code>HSI_VALUE</code>	Defines the value of the internal oscillator (HSI) expressed in Hz.	64 000 000 Hz
<code>LSE_VALUE</code>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz
<code>LSE_STARTUP_TIMEOUT</code>	Timeout for LSE start-up, expressed in ms	5000
<code>VDD_VALUE</code>	VDD value	3300 (mV)
<code>USERTOS</code>	Enables the use of RTOS	FALSE (reserved for future use)

Note: The `stm32h7xx_hal_conf_template.h` file is located in the HAL driver Inc folder. It should be copied to the user folder, renamed and modified as described above.

Note: By default, the values defined in the `stm32h7xx_hal_conf_template.h` file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

3.11 HAL system peripheral handling

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

3.11.1 Clock

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

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct). This function configures/enables multiple clock sources (HSE, HSI, LSE, LSI, PLL).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
 - selects the system clock source
 - configures AHB, APB1, APB2, APB3 and APB4 clock dividers
 - configures the number of Flash memory wait states
 - updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (RTC, USB...). In this case, the clock configuration is performed by an extended API defined in `stm32h7xx_hal_rcc_ex.c`:
`HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)`.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() Clock de-init function that return clock configuration to reset state
- Get clock functions that allow retrieving various clock configurations (system clock, HCLK, PCLK1, PCLK2, ...)
- MCO and CSS configuration functions

A set of macros are defined in `stm32h7xx_hal_rcc.h` and `stm32h7xx_hal_rcc_ex.h`. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- __PPP_CLK_ENABLE/__PPP_CLK_DISABLE to enable/disable the peripheral clock
- __PPP_FORCE_RESET/__PPP_RELEASE_RESET to force/release peripheral reset
- __PPP_CLK_SLEEP_ENABLE/__PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during low power (Sleep) mode.

3.11.2 GPIOs

GPIO HAL APIs are the following:

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

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

When selecting EXTI mode with interrupt generation, the user must call `HAL_GPIO_EXTI_IRQHandler()` from `stm32h7xx_it.c` and implement `HAL_GPIO_EXTI_Callback()`

The table below describes the `GPIO_InitTypeDef` structure field.

Table 12. Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]

Structure field	Description
Mode	<p>Specifies the operating mode for the selected pins: GPIO mode or EXTI mode.</p> <p>Possible values are:</p> <ul style="list-style-type: none">• <u>GPIO mode</u><ul style="list-style-type: none">– GPIO_MODE_INPUT : Input floating– GPIO_MODE_OUTPUT_PP : Output push-pull– GPIO_MODE_OUTPUT_OD : Output open drain– GPIO_MODE_AF_PP : Alternate function push-pull– GPIO_MODE_AF_OD : Alternate function open drain– GPIO_MODE_ANALOG : Analog mode• <u>External Interrupt mode</u><ul style="list-style-type: none">– GPIO_MODE_IT_RISING : Rising edge trigger detection– GPIO_MODE_IT_FALLING : Falling edge trigger detection– GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection• <u>External Event mode</u><ul style="list-style-type: none">– GPIO_MODE_EVT_RISING : Rising edge trigger detection– GPIO_MODE_EVT_FALLING : Falling edge trigger detection– GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	<p>Specifies the Pull-up or Pull-down activation for the selected pins.</p> <p>Possible values are:</p> <p>GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN</p>
Speed	<p>Specifies the speed for the selected pins</p> <p>Possible values are:</p> <p>GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH</p>

Please find below typical GPIO configuration examples:

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

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

- Configuring PA0 as external interrupt with falling edge sensitivity:

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

3.11.3

Cortex NVIC and SysTick timer

The Cortex HAL driver, `stm32h7xx_hal_cortex.c`, provides APIs to handle NVIC and SysTick. The supported APIs include:

- `HAL_NVIC_SetPriority()`/ `HAL_NVIC_SetPriorityGrouping()`

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

3.11.4 PWR

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

- PVD configuration, enabling/disabling and interrupt handling
 - HAL_PWR_ConfigPVD()
 - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
 - HAL_PWR_PVD_IRQHandler()
 - HAL_PWR_PVDCallback()
- Wakeup pin configuration
 - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
- Low-power mode entry
 - HAL_PWR_EnterSLEEPMode()
 - HAL_PWR_EnterSTOPMode()
 - HAL_PWR_EnterSTANDBYMode()

3.11.5 EXTI

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

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

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

Table 13. Description of EXTI configuration macros

Macros	Description
<code>_HAL_PPP_{SUBLOCK}__EXTI_ENABLE_IT()</code>	Enables a given EXTI line interrupt Example: <code>_HAL_PWR_PVD_EXTI_ENABLE_IT()</code>
<code>_HAL_PPP_{SUBLOCK}__EXTI_DISABLE_IT()</code>	Disables a given EXTI line. Example: <code>_HAL_PWR_PVD_EXTI_DISABLE_IT()</code>
<code>_HAL_PPP_{SUBLOCK}__EXTI_GET_FLAG()</code>	Gets a given EXTI line interrupt flag pending bit status. Example: <code>_HAL_PWR_PVD_EXTI_GET_FLAG()</code>

Macros	Description
<code>_HAL_PPP_{SUBBLOCK}_EXTI_CLEAR_FLAG()</code>	Clears a given EXTI line interrupt flag pending bit. Example: <code>_HAL_PWR_PVD_EXTI_CLEAR_FLAG()</code>
<code>_HAL_PPP_{SUBBLOCK}_EXTI_GENERATE_SWIT()</code>	Generates a software interrupt for a given EXTI line. Example: <code>_HAL_PWR_PVD_EXTI_GENERATE_SWIT()</code>
<code>_HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()</code>	Enables a given EXTI line event Example: <code>_HAL_RTC_WAKEUP_EXTI_ENABLE_EVENT()</code>
<code>_HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()</code>	Disables a given EXTI line event Example: <code>_HAL_RTC_WAKEUP_EXTI_DISABLE_EVENT()</code>
<code>_HAL_PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()</code>	Configures an EXTI Interrupt or Event on rising edge
<code>_HAL_PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()</code>	Enables an EXTI Interrupt or Event on Falling edge
<code>_HAL_PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()</code>	Disables an EXTI Interrupt or Event on rising edge
<code>_HAL_PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()</code>	Disables an EXTI Interrupt or Event on Falling edge
<code>_HAL_PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()</code>	Enables an EXTI Interrupt or Event on Rising/Falling edge
<code>_HAL_PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING_EDGE()</code>	Disables an EXTI Interrupt or Event on Rising/Falling edge

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32h7xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback()).

3.11.6 DMA

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

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

- Transfer Direction
- Source and Destination data formats
- Circular, Normal or peripheral flow control mode
- Channels Priority level
- Source and Destination Increment mode

Two operating modes are available:

- Polling mode I/O operation
 1. Use HAL_DMA_Start() to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
 2. Use HAL_DMA_PollForTransfer() to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
 1. Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
 2. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()

3. Use HAL_DMA_Start_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case the DMA interrupt is configured.
4. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
5. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (i.e. a member of DMA handle structure).

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

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

The most used DMA HAL driver macros are the following:

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

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

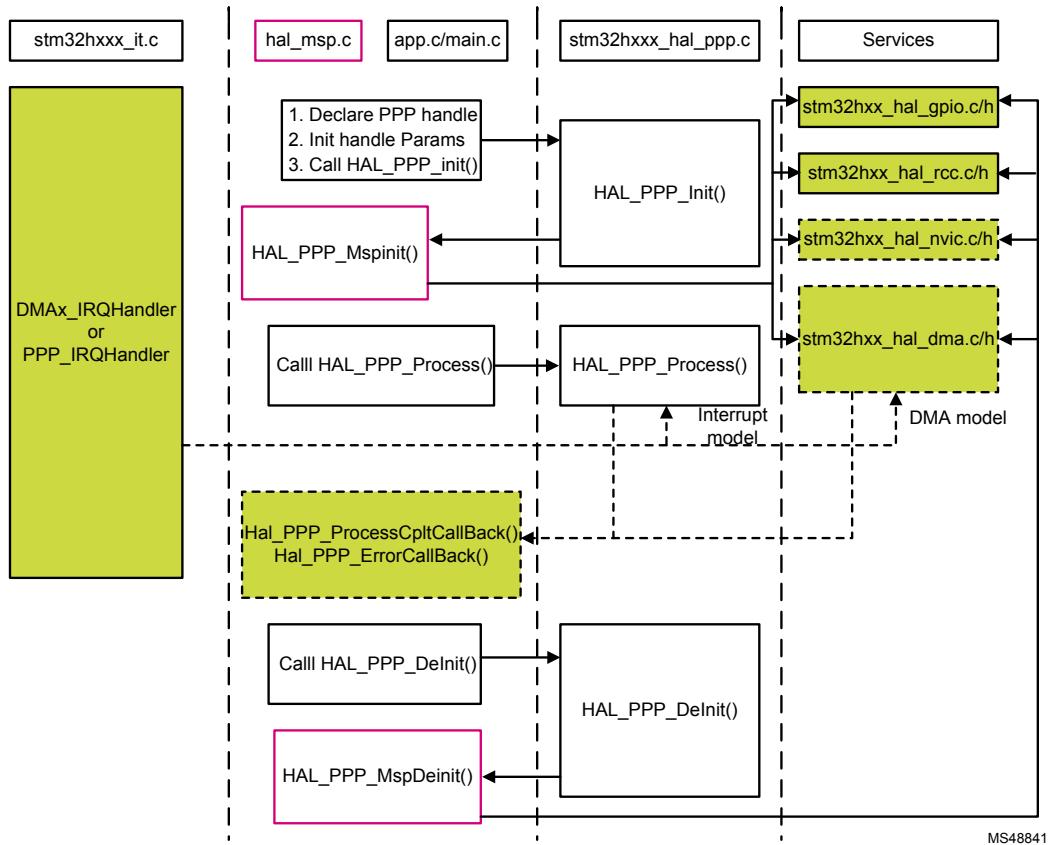
Note: *DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.*

3.12 How to use HAL drivers

3.12.1 HAL usage models

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

Figure 6. HAL driver model

**Note:**

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

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

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

3.12.2 HAL initialization**3.12.2.1 HAL global initialization**

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file `stm32h7xx_hal.c`.

- `HAL_Init()`: this function must be called at application startup to
 - initialize data/instruction cache and pre-fetch queue
 - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
 - call `HAL_MspInit()` user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). `HAL_MspInit()` is defined as “weak” empty function in the HAL drivers.
- `HAL_Delinit()`
 - resets all peripherals
 - calls function `HAL_MspDelinit()` which is user callback function to do system level De-Initializations.
- `HAL_GetTick()`: this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.

- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer. Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

3.12.2.2 System clock initialization

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in his own code. Please find below the typical clock configuration sequence:

```
static void SystemClock_Config(void)
{
    RCC_ClkInitTypeDef RCC_ClkInitStruct;
    RCC_OscInitTypeDef RCC_OscInitStruct;
    HAL_StatusTypeDef ret = HAL_OK;

    /*!< Supply configuration update enable */
    MODIFY_REG(PWR->CR3, PWR_CR3_SCUEN, 0);

    /* The voltage scaling allows optimizing the power consumption when the device is
       clocked below the maximum system frequency, to update the voltage scaling value
       regarding system frequency refer to product datasheet. */
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}

    /* Enable HSE Oscillator and activate PLL with HSE as source */
    RCC_OscInitStruct.OscillatoreType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSISState = RCC_HSI_OFF;
    RCC_OscInitStruct.CSISState = RCC_CSI_OFF;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

    RCC_OscInitStruct.PLL.PLLM = 5;
    RCC_OscInitStruct.PLL.PLLN = 160;
    RCC_OscInitStruct.PLL.PLLP = 2;
    RCC_OscInitStruct.PLL.PLLR = 2;
    RCC_OscInitStruct.PLL.PLLQ = 4;

    RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
    RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_2;
    ret = HAL_RCC_OscConfig(&RCC_OscInitStruct);
    if(ret != HAL_OK)
    {
        Error_Handler();
    }

    /* Select PLL as system clock source and configure bus clocks dividers */
    RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_D1PCLK1 | RCC_CLOCKTYPE_PCLK1 | \
                                  RCC_CLOCKTYPE_PCLK2 | RCC_CLOCKTYPE_D3PCLK1);

    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV2;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV2;
```

```
RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV2;
RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV2;
ret = HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4);
if(ret != HAL_OK)
{
    Error_Handler();
}

/*
Note : The activation of the I/O Compensation Cell is recommended with communication interfaces
        (GPIO, SPI, FMC, QSPI ...) when operating at high frequencies (please refer to product datasheet)
        The I/O Compensation Cell activation procedure requires :
        - The activation of the CSI clock
        - The activation of the SYSCFG clock
        - Enabling the I/O Compensation Cell : setting bit[0] of register SYSCFG_CCSR

        To do this please uncomment the following code
*/
/* __HAL_RCC_CSI_ENABLE() ;
   __HAL_RCC_SYSCFG_CLK_ENABLE() ;
HAL_EnableCompensationCell();
*/}
```

3.12.2.3 HAL MSP initialization process

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

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

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

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

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

The HAL MSP callback is implemented inside the *stm32h7xx_hal_msp.c* file in the user folders. An *stm32h7xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32h7xx_hal_msp.c file contains the following functions:

Table 14. MSP functions

Routine	Description
<code>void HAL_MspInit()</code>	Global MSP initialization routine
<code>void HAL_MspDeInit()</code>	Global MSP de-initialization routine
<code>void HAL_PPP_MspInit()</code>	PPP MSP initialization routine
<code>void HAL_PPP_MspDeInit()</code>	PPP MSP de-initialization routine

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

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

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

3.12.3

HAL IO operation process

The HAL functions with internal data processing like Transmit, Receive, Write and Read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode
- DMA mode

3.12.3.1

Polling mode

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

The example below shows the typical Polling mode processing sequence :

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_t Size, uint32_t Timeout)
{
if((pData == NULL ) || (Size == 0 ))
{
return HAL_ERROR;
}
(...) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
```

```
}
```

3.12.3.2 Interrupt mode

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

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

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

To use a process in Interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and *HAL_PPP_IRQHandler* in *stm32h7xx_it.c*.

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

An example of use is illustrated below:

main.c file:

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

stm32h7xx_it.c file:

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

3.12.3.3 DMA mode

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

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

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32h7xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

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

The initialization is done as follows (UART example):

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

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

An example of use is illustrated below:

main.c file:

```
USART_HandleTypeDef UartHandle;
int main(void)
```

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

stm32h7xx_it.c file:

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

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

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

3.12.4 Timeout and error management

3.12.4.1 Timeout management

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

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

The timeout possible value are the following:

Table 15. Timeout values

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

1. *HAL_MAX_DELAY* is defined in the *stm32h7xx_hal_def.h* as 0xFFFFFFFF

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

Example of fixed timeout:

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

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

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
(
...)
timeout = HAL_GetTick() + Timeout;
(
...)
while(ProcessOngoing)
{
(
...)
if(Timeout != HAL_MAX_DELAY)
{
if(HAL_GetTick() >= timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return hppp->State;
}
}
(
...)
}
```

3.12.4.2 Error management

The HAL drivers implement a check on the following items:

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

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pData, ui
nt32 Size) { if ((pData == NULL) || (Size == 0)) { return HAL_ERROR; } }
```

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

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

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

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

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

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

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

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

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

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

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

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

3.12.4.3

Run-time checking

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

To enable the run-time checking, use the *assert_param* macro, and leave the define **USE_FULL_ASSERT** uncommented in *stm32h7xx_hal_conf.h* file.

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

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

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

The `assert_param` macro is implemented in `stm32h7xx_hal_conf.h`:

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

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

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

Note:

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

4 HAL System Driver

4.1 HAL Firmware driver API description

4.1.1 How to use this driver

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

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

4.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [**HAL_Init**](#)
- [**HAL_DeInit**](#)
- [**HAL_MspInit**](#)
- [**HAL_MspDeInit**](#)
- [**HAL_InitTick**](#)

4.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier
- Enable/Disable Debug module during SLEEP mode

- Enable/Disable Debug module during STOP mode
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:

- [HAL_IncTick](#)
- [HAL_GetTick](#)
- [HAL_GetTickPrio](#)
- [HAL_SetTickFreq](#)
- [HAL_GetTickFreq](#)
- [HAL_Delay](#)
- [HAL_SuspendTick](#)
- [HAL_ResumeTick](#)
- [HAL_GetHalVersion](#)
- [HAL_GetREVID](#)
- [HAL_GetDEVID](#)
- [HAL_SYSCFG_VREFBUF_VoltageScalingConfig](#)
- [HAL_SYSCFG_VREFBUF_HighImpedanceConfig](#)
- [HAL_SYSCFG_VREFBUF_TrimmingConfig](#)
- [HAL_SYSCFG_EnableVREFBUF](#)
- [HAL_SYSCFG_DisableVREFBUF](#)
- [HAL_SYSCFG_ETHInterfaceSelect](#)
- [HAL_SYSCFG_AnalogSwitchConfig](#)
- [HAL_SYSCFG_EnableBOOST](#)
- [HAL_SYSCFG_DisableBOOST](#)
- [HAL_SYSCFG_CM7BootAddConfig](#)
- [HAL_EnableCompensationCell](#)
- [HAL_DisableCompensationCell](#)
- [HAL_SYSCFG_EnableIOSpeedOptimize](#)
- [HAL_SYSCFG_DisableIOSpeedOptimize](#)
- [HAL_SYSCFG_CompensationCodeSelect](#)
- [HAL_SYSCFG_CompensationCodeConfig](#)
- [HAL_EnableDBGSleepMode](#)
- [HAL_DisableDBGSleepMode](#)
- [HAL_EnableDBGStopMode](#)
- [HAL_DisableDBGStopMode](#)
- [HAL_EnableDBGStandbyMode](#)
- [HAL_DisableDBGStandbyMode](#)
- [HAL_EnableDomain3DBGStopMode](#)
- [HAL_DisableDomain3DBGStopMode](#)
- [HAL_EnableDomain3DBGStandbyMode](#)
- [HAL_DisableDomain3DBGStandbyMode](#)
- [HAL_SetFMCMemorySwappingConfig](#)
- [HAL_GetFMCMemorySwappingConfig](#)
- [HAL_EXTI_EdgeConfig](#)
- [HAL_EXTI_GenerateSWInterrupt](#)
- [HAL_EXTI_D1_ClearFlag](#)
- [HAL_EXTI_D1_EventInputConfig](#)
- [HAL_EXTI_D3_EventInputConfig](#)
- [HAL_Delay](#)

4.1.4 Detailed description of functions

HAL_Init

Function name

HAL_StatusTypeDef HAL_Init (void)

Function description

This function is used to initialize the HAL Library; it must be the first instruction to be executed in the main program (before to call any other HAL function), it performs the following: Configures the SysTick to generate an interrupt each 1 millisecond, which is clocked by the HSI (at this stage, the clock is not yet configured and thus the system is running from the internal HSI at 16 MHz).

Return values

- **HAL:** status

Notes

- SysTick is used as time base for the HAL_Delay() function, the application need to ensure that the SysTick time base is always set to 1 millisecond to have correct HAL operation.

HAL_DeInit

Function name

HAL_StatusTypeDef HAL_DeInit (void)

Function description

This function de-Initializes common part of the HAL and stops the systick.

Return values

- **HAL:** status

HAL_MspInit

Function name

void HAL_MspInit (void)

Function description

Initializes the MSP.

Return values

- **None:**

HAL_MspDeInit

Function name

void HAL_MspDeInit (void)

Function description

DeInitializes the MSP.

Return values

- **None:**

HAL_InitTick

Function name

HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)

Function description

This function configures the source of the time base.

Parameters

- **TickPriority:** Tick interrupt priority.

Return values

- **HAL:** status

Notes

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

HAL_IncTick

Function name

```
void HAL_IncTick (void )
```

Function description

This function is called to increment a global variable "uwTick" used as application time base.

Return values

- **None:**

Notes

- In the default implementation, this variable is incremented each 1ms in Systick ISR.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_Delay

Function name

```
void HAL_Delay ( __IO uint32_t Delay)
```

Function description

HAL_GetTick

Function name

```
uint32_t HAL_GetTick (void )
```

Function description

Provides a tick value in millisecond.

Return values

- **tick:** value

Notes

- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTickPrio

Function name

`uint32_t HAL_GetTickPrio (void)`

Function description

This function returns a tick priority.

Return values

- **tick:** priority

HAL_SetTickFreq

Function name

`HAL_StatusTypeDef HAL_SetTickFreq (HAL_TickFreqTypeDef Freq)`

Function description

Set new tick Freq.

Return values

- **Status:**

HAL_GetTickFreq

Function name

`HAL_TickFreqTypeDef HAL_GetTickFreq (void)`

Function description

Return tick frequency.

Return values

- **tick:** period in Hz

HAL_SuspendTick

Function name

`void HAL_SuspendTick (void)`

Function description

Suspend Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the SysTick interrupt will be disabled and so Tick increment is suspended.
- This function is declared as `__weak` to be overwritten in case of other implementations in user file.

HAL_ResumeTick

Function name

`void HAL_ResumeTick (void)`

Function description

Resume Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the SysTick interrupt will be enabled and so Tick increment is resumed.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetHalVersion

Function name

`uint32_t HAL_GetHalVersion (void)`

Function description

Returns the HAL revision.

Return values

- **version:** : 0xXYZR (8bits for each decimal, R for RC)

HAL_GetREVID

Function name

`uint32_t HAL_GetREVID (void)`

Function description

Returns the device revision identifier.

Return values

- **Device:** revision identifier

HAL_GetDEVID

Function name

`uint32_t HAL_GetDEVID (void)`

Function description

Returns the device identifier.

Return values

- **Device:** identifier

HAL_SYSCFG_ETHInterfaceSelect

Function name

`void HAL_SYSCFG_ETHInterfaceSelect (uint32_t SYSCFG_ETHInterface)`

Function description

Ethernet PHY Interface Selection either MII or RMII.

Parameters

- **SYSCFG_ETHInterface:** Selects the Ethernet PHY interface This parameter can be one of the following values:
 - SYSCFG_ETH_MII : Select the Media Independent Interface

- SYSCFG_ETH_RMII: Select the Reduced Media Independent Interface

Return values

- **None:**

HAL_SYSCFG_AnalogSwitchConfig

Function name

```
void HAL_SYSCFG_AnalogSwitchConfig (uint32_t SYSCFG_AnalogSwitch, uint32_t  
SYSCFG_SwitchState)
```

Function description

Analog Switch control for dual analog pads.

Parameters

- **SYSCFG_AnalogSwitch:** Selects the analog pad This parameter can be one or a combination of the following values:
 - SYSCFG_SWITCH_PA0 : Select PA0 analog switch
 - SYSCFG_SWITCH_PA1: Select PA1 analog switch
 - SYSCFG_SWITCH_PC2 : Select PC2 analog switch
 - SYSCFG_SWITCH_PC3: Select PC3 analog switch
- **SYSCFG_SwitchState:** Open or Close the analog switch between dual pads (PXn and PXn_C) This parameter can be one or a combination of the following values:
 - SYSCFG_SWITCH_PA0_OPEN
 - SYSCFG_SWITCH_PA0_CLOSE
 - SYSCFG_SWITCH_PA1_OPEN
 - SYSCFG_SWITCH_PA1_CLOSE
 - SYSCFG_SWITCH_PC2_OPEN
 - SYSCFG_SWITCH_PC2_CLOSE
 - SYSCFG_SWITCH_PC3_OPEN
 - SYSCFG_SWITCH_PC3_CLOSE

Return values

- **None:**

HAL_SYSCFG_EnableBOOST

Function name

```
void HAL_SYSCFG_EnableBOOST (void )
```

Function description

Enables the booster to reduce the total harmonic distortion of the analog switch when the supply voltage is lower than 2.7 V.

Return values

- **None:**

Notes

- Activating the booster allows to guaranty the analog switch AC performance when the supply voltage is below 2.7 V: in this case, the analog switch performance is the same on the full voltage range

HAL_SYSCFG_DisableBOOST

Function name

```
void HAL_SYSCFG_DisableBOOST (void )
```

Function description

Disables the booster.

Return values

- **None:**

Notes

- Activating the booster allows to guaranty the analog switch AC performance when the supply voltage is below 2.7 V: in this case, the analog switch performance is the same on the full voltage range

HAL_SYSCFG_CM7BootAddConfig

Function name

```
void HAL_SYSCFG_CM7BootAddConfig (uint32_t BootRegister, uint32_t BootAddress)
```

Function description

BootCM7 address 0 configuration.

Parameters

- **BootRegister:** :Specifies the Boot Address register (Address0 or Address1) This parameter can be one of the following values:
 - SYSCFG_BOOT_ADDR0 : Select the boot address0
 - SYSCFG_BOOT_ADDR1: Select the boot address1
- **BootAddress:** :Specifies the CM7 Boot Address to be loaded in Address0 or Address1

Return values

- **None:**

HAL_EnableCompensationCell

Function name

```
void HAL_EnableCompensationCell (void )
```

Function description

Enables the I/O Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.

HAL_DisableCompensationCell

Function name

```
void HAL_DisableCompensationCell (void )
```

Function description

Power-down the I/O Compensation Cell.

Return values

- **None:**

Notes

- The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.

HAL_SYSCFG_EnableIOSpeedOptimize

Function name

```
void HAL_SYSCFG_EnableIOSpeedOptimize (void )
```

Function description

To Enable optimize the I/O speed when the product voltage is low.

Return values

- **None:**

Notes

- This bit is active only if PRODUCT_BELOW_25V user option bit is set. It must be used only if the product supply voltage is below 2.5 V. Setting this bit when VDD is higher than 2.5 V might be destructive.

HAL_SYSCFG_DisableIOSpeedOptimize

Function name

```
void HAL_SYSCFG_DisableIOSpeedOptimize (void )
```

Function description

To Disable optimize the I/O speed when the product voltage is low.

Return values

- **None:**

Notes

- This bit is active only if PRODUCT_BELOW_25V user option bit is set. It must be used only if the product supply voltage is below 2.5 V. Setting this bit when VDD is higher than 2.5 V might be destructive.

HAL_SYSCFG_CompensationCodeSelect

Function name

```
void HAL_SYSCFG_CompensationCodeSelect (uint32_t SYSCFG_CompCode)
```

Function description

Code selection for the I/O Compensation cell.

Parameters

- **SYSCFG_CompCode:** Selects the code to be applied for the I/O compensation cell. This parameter can be one of the following values:
 - SYSCFG_CELL_CODE : Select Code from the cell (available in the SYSCFG_CCVR)
 - SYSCFG_REGISTER_CODE: Select Code from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Return values

- **None:**

HAL_SYSCFG_CompensationCodeConfig

Function name

```
void HAL_SYSCFG_CompensationCodeConfig (uint32_t SYSCFG_PMOSCode, uint32_t SYSCFG_NMOSCode)
```

Function description

Code selection for the I/O Compensation cell.

Parameters

- **SYSCFG_PMOSCode:** PMOS compensation code This code is applied to the I/O compensation cell when the CS bit of the SYSCFG_CMPCR is set
- **SYSCFG_NMOSCode:** NMOS compensation code This code is applied to the I/O compensation cell when the CS bit of the SYSCFG_CMPCR is set

Return values

- **None:**

HAL_EnableDBGSleepMode

Function name

void HAL_EnableDBGSleepMode (void)

Function description

Enable the Debug Module during Domain1 SLEEP mode.

Return values

- **None:**

HAL_DisableDBGSleepMode

Function name

void HAL_DisableDBGSleepMode (void)

Function description

Disable the Debug Module during Domain1 SLEEP mode.

Return values

- **None:**

HAL_EnableDBGStopMode

Function name

void HAL_EnableDBGStopMode (void)

Function description

Enable the Debug Module during Domain1 STOP mode.

Return values

- **None:**

HAL_DisableDBGStopMode

Function name

void HAL_DisableDBGStopMode (void)

Function description

Disable the Debug Module during Domain1 STOP mode.

Return values

- **None:**

HAL_EnableDBGStandbyMode

Function name

void HAL_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during Domain1 STANDBY mode.

Return values

- **None:**

HAL_DisableDBGStandbyMode

Function name

void HAL_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during Domain1 STANDBY mode.

Return values

- **None:**

HAL_EnableDomain3DBGStopMode

Function name

void HAL_EnableDomain3DBGStopMode (void)

Function description

Enable the Debug Module during Domain3 STOP mode.

Return values

- **None:**

HAL_DisableDomain3DBGStopMode

Function name

void HAL_DisableDomain3DBGStopMode (void)

Function description

Disable the Debug Module during Domain3 STOP mode.

Return values

- **None:**

HAL_EnableDomain3DBGStandbyMode

Function name

void HAL_EnableDomain3DBGStandbyMode (void)

Function description

Enable the Debug Module during Domain3 STANDBY mode.

Return values

- **None:**

HAL_DisableDomain3DBGStandbyMode

Function name

void HAL_DisableDomain3DBGStandbyMode (void)

Function description

Disable the Debug Module during Domain3 STANDBY mode.

Return values

- **None:**

HAL_EXTI_EdgeConfig

Function name

void HAL_EXTI_EdgeConfig (uint32_t EXTI_Line, uint32_t EXTI_Edge)

Function description

Configure the EXTI input event line edge.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0...EXTI_LINE88)excluding :line45, line81,line83 which are reserved
- **EXTI_Edge:** Specifies EXTI line Edge used. This parameter can be one of the following values :
 - EXTI_RISING_EDGE : Configurable line, with Rising edge trigger detection
 - EXTI_FALLING_EDGE: Configurable line, with Falling edge trigger detection

Return values

- **None:**

Notes

- No edge configuration for direct lines but for configurable lines:(EXTI_LINE0..EXTI_LINE21), EXTI_LINE49,EXTI_LINE51,EXTI_LINE82,EXTI_LINE84,EXTI_LINE85 and EXTI_LINE86.

HAL_EXTI_GenerateSWInterrupt

Function name

void HAL_EXTI_GenerateSWInterrupt (uint32_t EXTI_Line)

Function description

Generates a Software interrupt on selected EXTI line.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0..EXTI_LINE21),EXTI_LINE49,EXTI_LINE51,EXTI_LINE82,EXTI_LINE84,EXTI_LINE85 and EXTI_LINE86.

Return values

- **None:**

HAL_EXTI_D1_ClearFlag

Function name

void HAL_EXTI_D1_ClearFlag (uint32_t EXTI_Line)

Function description

Clears the EXTI's line pending flags for Domain D1.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0...EXTI_LINE88)excluding :line45, line81,line83 which are reserved

Return values

- **None:**

HAL_EXTI_D1_EventInputConfig

Function name

```
void HAL_EXTI_D1_EventInputConfig (uint32_t EXTI_Line, uint32_t EXTI_Mode, uint32_t EXTI_LineCmd)
```

Function description

Configure the EXTI input event line for Domain D1.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0...EXTI_LINE88)excluding :line45, line81,line83 which are reserved
- **EXTI_Mode:** Specifies which EXTI line is used as interrupt or an event. This parameter can be one or a combination of the following values :
 - EXTI_MODE_IT : Interrupt Mode selected
 - EXTI_MODE_EVT : Event Mode selected
- **EXTI_LineCmd:** controls (Enable/Disable) the EXTI line.

Return values

- **None:**

HAL_EXTI_D3_EventInputConfig

Function name

```
void HAL_EXTI_D3_EventInputConfig (uint32_t EXTI_Line, uint32_t EXTI_LineCmd, uint32_t  
EXTI_ClearSrc)
```

Function description

Configure the EXTI input event line for Domain D3.

Parameters

- **EXTI_Line:** Specifies the EXTI LINE, it can be one of the following values, (EXTI_LINE0...EXTI_LINE15), (EXTI_LINE19...EXTI_LINE21),EXTI_LINE25, EXTI_LINE34, EXTI_LINE35,EXTI_LINE41, (EXTI_LINE48...EXTI_LINE53),EXTI_LINE88
- **EXTI_LineCmd:** controls (Enable/Disable) the EXTI line.
- **EXTI_ClearSrc:** Specifies the clear source of D3 pending event. This parameter can be one of the following values :
 - BDMA_CH6_CLEAR : BDMA ch6 event selected as D3 domain pendclear source
 - BDMA_CH7_CLEAR : BDMA ch7 event selected as D3 domain pendclear source
 - LPTIM4_OUT_CLEAR : LPTIM4 out selected as D3 domain pendclear source
 - LPTIM5_OUT_CLEAR : LPTIM5 out selected as D3 domain pendclear source

Return values

- **None:**

HAL_SetFMCMemorySwappingConfig

Function name

```
void HAL_SetFMCMemorySwappingConfig (uint32_t BankMapConfig)
```

Function description

Set the FMC Memory Mapping Swapping config.

Parameters

- **BankMapConfig:** Defines the FMC Bank mapping configuration. This parameter can be FMC_SWAPBMAP_DISABLE, FMC_SWAPBMAP_SDRAM_SRAM, FMC_SWAPBMAP_SDRAMB2

Return values

- **HAL:** state

HAL_GetFMCMemorySwappingConfig

Function name

```
uint32_t HAL_GetFMCMemorySwappingConfig (void )
```

Function description

Get FMC Bank mapping mode.

Return values

- **The:** FMC Bank mapping mode. This parameter can be FMC_SWAPBMAP_DISABLE, FMC_SWAPBMAP_SDRAM_SRAM, FMC_SWAPBMAP_SDRAMB2

HAL_SYSCFG_VREFBUF_VoltageScalingConfig

Function name

```
void HAL_SYSCFG_VREFBUF_VoltageScalingConfig (uint32_t VoltageScaling)
```

Function description

Configure the internal voltage reference buffer voltage scale.

Parameters

- **VoltageScaling:** specifies the output voltage to achieve This parameter can be one of the following values:
 - SYSCFG_VREFBUF_VOLTAGE_SCALE0: VREF_OUT1 around 2.048 V. This requires VDDA equal to or higher than 2.4 V.
 - SYSCFG_VREFBUF_VOLTAGE_SCALE1: VREF_OUT2 around 2.5 V. This requires VDDA equal to or higher than 2.8 V.
 - SYSCFG_VREFBUF_VOLTAGE_SCALE2: VREF_OUT3 around 1.5 V. This requires VDDA equal to or higher than 1.8 V.
 - SYSCFG_VREFBUF_VOLTAGE_SCALE3: VREF_OUT4 around 1.8 V. This requires VDDA equal to or higher than 2.1 V.

Return values

- **None:**

HAL_SYSCFG_VREFBUF_HighImpedanceConfig

Function name

```
void HAL_SYSCFG_VREFBUF_HighImpedanceConfig (uint32_t Mode)
```

Function description

Configure the internal voltage reference buffer high impedance mode.

Parameters

- **Mode:** specifies the high impedance mode This parameter can be one of the following values:
 - SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE: VREF+ pin is internally connect to VREFINT output.
 - SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE: VREF+ pin is high impedance.

Return values

- **None:**

HAL_SYSCFG_VREFBUF_TrimmingConfig

Function name

```
void HAL_SYSCFG_VREFBUF_TrimmingConfig (uint32_t TrimmingValue)
```

Function description

Tune the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

HAL_SYSCFG_EnableVREFBUF

Function name

```
HAL_StatusTypeDef HAL_SYSCFG_EnableVREFBUF (void )
```

Function description

Enable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **HAL_OK/HAL_TIMEOUT:**

HAL_SYSCFG_DisableVREFBUF

Function name

```
void HAL_SYSCFG_DisableVREFBUF (void )
```

Function description

Disable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

4.2 HAL Firmware driver defines

4.2.1 HAL

HAL Exported Macros

`_HAL_DBGMCU_FREEZE_WWDG1`

`_HAL_DBGMCU_FREEZE_TIM2`

`_HAL_DBGMCU_FREEZE_TIM3`

`_HAL_DBGMCU_FREEZE_TIM4`

`_HAL_DBGMCU_FREEZE_TIM5`

`_HAL_DBGMCU_FREEZE_TIM6`

`_HAL_DBGMCU_FREEZE_TIM7`

`_HAL_DBGMCU_FREEZE_TIM12`

_HAL_DBGMCU_FREEZE_TIM13

_HAL_DBGMCU_FREEZE_TIM14

_HAL_DBGMCU_FREEZE_LPTIM1

_HAL_DBGMCU_FREEZE_I2C1

_HAL_DBGMCU_FREEZE_I2C2

_HAL_DBGMCU_FREEZE_I2C3

_HAL_DBGMCU_FREEZE_FDCAN

_HAL_DBGMCU_FREEZE_TIM1

_HAL_DBGMCU_FREEZE_TIM8

_HAL_DBGMCU_FREEZE_TIM15

_HAL_DBGMCU_FREEZE_TIM16

_HAL_DBGMCU_FREEZE_TIM17

_HAL_DBGMCU_FREEZE_HRTIM

_HAL_DBGMCU_FREEZE_I2C4

_HAL_DBGMCU_FREEZE_LPTIM2

_HAL_DBGMCU_FREEZE_LPTIM3

_HAL_DBGMCU_FREEZE_LPTIM4

_HAL_DBGMCU_FREEZE_LPTIM5

_HAL_DBGMCU_FREEZE_RTC

_HAL_DBGMCU_FREEZE_IWDG1

_HAL_DBGMCU_UnFreeze_WWDG1

_HAL_DBGMCU_UnFreeze_TIM2

_HAL_DBGMCU_UnFreeze_TIM3

_HAL_DBGMCU_UnFreeze_TIM4

_HAL_DBGMCU_UnFreeze_TIM5

_HAL_DBGMCU_UnFreeze_TIM6

_HAL_DBGMCU_UnFreeze_TIM7

_HAL_DBGMCU_UnFreeze_TIM12

`_HAL_DBGMCU_UnFreeze_TIM13`
`_HAL_DBGMCU_UnFreeze_TIM14`
`_HAL_DBGMCU_UnFreeze_LPTIM1`
`_HAL_DBGMCU_UnFreeze_I2C1`
`_HAL_DBGMCU_UnFreeze_I2C2`
`_HAL_DBGMCU_UnFreeze_I2C3`
`_HAL_DBGMCU_UnFreeze_FDCAN`
`_HAL_DBGMCU_UnFreeze_TIM1`
`_HAL_DBGMCU_UnFreeze_TIM8`
`_HAL_DBGMCU_UnFreeze_TIM15`
`_HAL_DBGMCU_UnFreeze_TIM16`
`_HAL_DBGMCU_UnFreeze_TIM17`
`_HAL_DBGMCU_UnFreeze_HRTIM`
`_HAL_DBGMCU_UnFreeze_I2C4`
`_HAL_DBGMCU_UnFreeze_LPTIM2`
`_HAL_DBGMCU_UnFreeze_LPTIM3`
`_HAL_DBGMCU_UnFreeze_LPTIM4`
`_HAL_DBGMCU_UnFreeze_LPTIM5`
`_HAL_DBGMCU_UnFreeze_RTC`
`_HAL_DBGMCU_UnFreeze_IWDG1`

Event Input Config

`EXTI_MODE_IT`
`EXTI_MODE_EVT`
`EXTI_RISING_EDGE`
`EXTI_FALLING_EDGE`
`IS_EXTI_EDGE_LINE`
`IS_EXTI_MODE_LINE`
`EXTI_LINE0`

External interrupt LINE 0

EXTI_LINE1

External interrupt LINE 1

EXTI_LINE2

External interrupt LINE 2

EXTI_LINE3

External interrupt LINE 3

EXTI_LINE4

External interrupt LINE 4

EXTI_LINE5

External interrupt LINE 5

EXTI_LINE6

External interrupt LINE 6

EXTI_LINE7

External interrupt LINE 7

EXTI_LINE8

External interrupt LINE 8

EXTI_LINE9

External interrupt LINE 9

EXTI_LINE10

External interrupt LINE 10

EXTI_LINE11

External interrupt LINE 11

EXTI_LINE12

External interrupt LINE 12

EXTI_LINE13

External interrupt LINE 13

EXTI_LINE14

External interrupt LINE 14

EXTI_LINE15

External interrupt LINE 15

EXTI_LINE16

EXTI_LINE17

EXTI_LINE18

EXTI_LINE19

EXTI_LINE20

EXTI_LINE21

EXTI_LINE22

EXTI_LINE23

EXTI_LINE24

EXTI_LINE25

EXTI_LINE26

EXTI_LINE27

EXTI_LINE28

EXTI_LINE29

EXTI_LINE30

EXTI_LINE31

EXTI_LINE32

EXTI_LINE33

EXTI_LINE34

EXTI_LINE35

EXTI_LINE36

EXTI_LINE37

EXTI_LINE38

EXTI_LINE39

EXTI_LINE40

EXTI_LINE41

EXTI_LINE42

EXTI_LINE43

EXTI_LINE44

EXTI_LINE47

EXTI_LINE48

EXTI_LINE49

EXTI_LINE50

EXTI_LINE51

EXTI_LINE52

EXTI_LINE53

EXTI_LINE54

EXTI_LINE55

EXTI_LINE56

EXTI_LINE57

EXTI_LINE58

EXTI_LINE59

EXTI_LINE60

EXTI_LINE61

EXTI_LINE62

EXTI_LINE63

EXTI_LINE64

EXTI_LINE65

EXTI_LINE66

EXTI_LINE67

EXTI_LINE68

EXTI_LINE69

EXTI_LINE70

EXTI_LINE71

EXTI_LINE72

EXTI_LINE73

EXTI_LINE74

EXTI_LINE75

EXTI_LINE76

EXTI_LINE85

EXTI_LINE86

EXTI_LINE87

IS EXTI_CONFIG_LINE

IS EXTI_ALL_LINE

IS EXTI_D1_LINE

IS EXTI_D3_LINE

BDMA_CH6_CLEAR

BDMA ch6 event selected as D3 domain pendclear source

BDMA_CH7_CLEAR

BDMA ch7 event selected as D3 domain pendclear source

LPTIM4_OUT_CLEAR

LPTIM4 out selected as D3 domain pendclear source

LPTIM5_OUT_CLEAR

LPTIM5 out selected as D3 domain pendclear source

IS EXTI_D3_CLEAR

SwapBankMapping Config

FMC_SWAPBMAP_DISABLE

FMC_SWAPBMAP_SDRAM_SRAM

FMC_SWAPBMAP_SDRAMB2

IS_FMC_SWAPBMAP_MODE

Analog Switch Config

SYSCFG_SWITCH_PA0

Select PA0 analog switch

SYSCFG_SWITCH_PA1

Select PA1 analog switch

SYSCFG_SWITCH_PC2

Select PC2 analog switch

SYSCFG_SWITCH_PC3

Select PC3 analog switch

IS_SYSCFG_ANALOG_SWITCH

SYSCFG_SWITCH_PA0_OPEN

PA0 analog switch opened

SYSCFG_SWITCH_PA0_CLOSE

PA0 analog switch closed

SYSCFG_SWITCH_PA1_OPEN

PA1 analog switch opened

SYSCFG_SWITCH_PA1_CLOSE

PA1 analog switch closed

SYSCFG_SWITCH_PC2_OPEN

PC2 analog switch opened

SYSCFG_SWITCH_PC2_CLOSE

PC2 analog switch closed

SYSCFG_SWITCH_PC3_OPEN

PC3 analog switch opened

SYSCFG_SWITCH_PC3_CLOSE

PC3 analog switch closed

IS_SYSCFG_SWITCH_STATE

Boot Config

SYSCFG_BOOT_ADDR0

Select Boot address0

SYSCFG_BOOT_ADDR1

Select Boot address1

IS_SYSCFG_BOOT_REGISTER

IS_SYSCFG_BOOT_ADDRESS

Ethernet Config

SYSCFG_ETH_MII

Select the Media Independent Interface

SYSCFG_ETH_RMII

Select the Reduced Media Independent Interface

IS_SYSCFG_ETHERNET_CONFIG

IOCompensationCell Config

SYSCFG_CELL_CODE

Select Code from the cell

SYSCFG_REGISTER_CODE

Code from the SYSCFG compensation cell code register

IS_SYSCFG_CODE_SELECT

IS_SYSCFG_CODE_CONFIG

VREFBUF High Impedance

SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE

VREF_plus pin is internally connected to Voltage reference buffer output

SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE

VREF_plus pin is high impedance

IS_SYSCFG_VREFBUF_HIGH_IMPEDANCE

IS_SYSCFG_VREFBUF_TRIMMING

VREFBUF Voltage Scale

SYSCFG_VREFBUF_VOLTAGE_SCALE0

Voltage reference scale 0 (VREF_OUT2)

SYSCFG_VREFBUF_VOLTAGE_SCALE1

Voltage reference scale 1 (VREF_OUT1)

SYSCFG_VREFBUF_VOLTAGE_SCALE2

Voltage reference scale 2 (VREF_OUT4)

SYSCFG_VREFBUF_VOLTAGE_SCALE3

Voltage reference scale 3 (VREF_OUT3)

IS_SYSCFG_VREFBUF_VOLTAGE_SCALE

5 HAL ADC Generic Driver

5.1 ADC Firmware driver registers structures

5.1.1 ADC_OversamplingTypeDef

Data Fields

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

Field Documentation

- *uint32_t ADC_OversamplingTypeDef::Ratio*

Configures the oversampling ratio.

- *uint32_t ADC_OversamplingTypeDef::RightBitShift*

Configures the division coefficient for the Oversampler. This parameter can be a value of **ADC Extended Oversampling Right Shift**

- *uint32_t ADC_OversamplingTypeDef::TriggeredMode*

Selects the regular triggered oversampling mode. This parameter can be a value of **ADC Extended Triggered Regular Oversampling**

- *uint32_t ADC_OversamplingTypeDef::OversamplingStopReset*

Selects the regular oversampling mode. The oversampling is either temporary stopped or reset upon an injected sequence interruption. If oversampling is enabled on both regular and injected groups, this parameter is discarded and forced to setting "ADC_REGOVERSAMPLING_RESUMED_MODE" (the oversampling buffer is zeroed during injection sequence). This parameter can be a value of **ADC Extended Regular Oversampling Continued or Resumed Mode**

5.1.2 ADC_InitTypeDef

Data Fields

- *uint32_t ClockPrescaler*
- *uint32_t Resolution*
- *uint32_t ScanConvMode*
- *uint32_t EOCSelection*
- *FunctionalState LowPowerAutoWait*
- *FunctionalState ContinuousConvMode*
- *uint32_t NbrOfConversion*
- *FunctionalState DiscontinuousConvMode*
- *uint32_t NbrOfDiscConversion*
- *uint32_t ExternalTrigConv*
- *uint32_t ExternalTrigConvEdge*
- *uint32_t ConversionDataManagement*
- *uint32_t Overrun*
- *uint32_t LeftBitShift*
- *FunctionalState BoostMode*
- *FunctionalState OversamplingMode*

- ***ADC_OversamplingTypeDef Oversampling***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ClockPrescaler***

Select ADC clock source (synchronous clock derived from APB clock or asynchronous clock derived from System/PLL2/PLL3 clocks) and clock prescaler. This parameter can be a value of ***ADC clock source and clock prescaler***. Note: The clock is common for all the ADCs. Note: In case of usage of channels on injected group, ADC frequency should be lower than AHB clock frequency /4 for resolution 16, 14, 12 or 10 bits, AHB clock frequency /3 for resolution 8 bits. Note: In case of synchronous clock mode based on HCLK/1, the configuration must be enabled only if the system clock has a 50% duty clock cycle (APB prescaler configured inside RCC must be bypassed and PCLK clock must have 50% duty cycle). Refer to reference manual for details. Note: In case of usage of the ADC dedicated PLL clock, it must be preliminarily enabled at RCC top level. Note: This parameter can be modified only if all ADCs are disabled.

- ***uint32_t ADC_InitTypeDef::Resolution***

Configure the ADC resolution. This parameter can be a value of ***ADC Resolution***

- ***uint32_t ADC_InitTypeDef::ScanConvMode***

Configure the sequencer of ADC groups regular and injected. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion' or 'InjectedNbrOfConversion' and rank of each channel in sequencer). Scan direction is upward: from rank 1 to rank 'n'. This parameter can be a value of ***ADC sequencer scan mode***

- ***uint32_t ADC_InitTypeDef::EOCSelection***

Specify which EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of unitary conversion or end of sequence conversions. This parameter can be a value of ***ADC sequencer end of unitary conversion or sequence conversions***.

- ***FunctionalState ADC_InitTypeDef::LowPowerAutoWait***

Select the dynamic low power Auto Delay: new conversion start only when the previous conversion (for ADC group regular) or previous sequence (for ADC group injected) has been retrieved by user software, using function ***HAL_ADC_GetValue()*** or ***HAL_ADCEx_InjectedGetValue()***. This feature automatically adapts the frequency of ADC conversions triggers to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: Do not use with interruption or DMA (***HAL_ADC_Start_IT()***, ***HAL_ADC_Start_DMA()***) since they clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion with ***HAL_ADC_Start()***, 2. Later on, when ADC conversion data is needed: use ***HAL_ADC_PollForConversion()*** to ensure that conversion is completed and ***HAL_ADC_GetValue()*** to retrieve conversion result and trig another conversion start. (in case of usage of ADC group injected, use the equivalent functions ***HAL_ADCExInjected_Start()***, ***HAL_ADCEx_InjectedGetValue()***, ...).

- ***FunctionalState ADC_InitTypeDef::ContinuousConvMode***

Specify whether the conversion is performed in single mode (one conversion) or continuous mode for ADC group regular, after the first ADC conversion start trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InitTypeDef::NbrOfConversion***

Specify the number of ranks that will be converted within the regular group sequencer. To use the regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 16. Note: This parameter must be modified when no conversion is on going on regular group (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- ***FunctionalState ADC_InitTypeDef::DiscontinuousConvMode***

Specify whether the conversions sequence of ADC group regular is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded.

Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.

- **`uint32_t ADC_InitTypeDef::NbrOfDiscConversion`**

Specifies the number of discontinuous conversions in which the main sequence of ADC group regular (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.

- **`uint32_t ADC_InitTypeDef::ExternalTrigConv`**

Select the external event source used to trigger ADC group regular conversion start. If set to ADC_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of **ADC group regular trigger source**. Caution: external trigger source is common to all ADC instances.

- **`uint32_t ADC_InitTypeDef::ExternalTrigConvEdge`**

Select the external event edge used to trigger ADC group regular conversion start. If trigger source is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of **ADC group regular trigger edge (when external trigger is selected)**

- **`uint32_t ADC_InitTypeDef::ConversionDataManagement`**

Specifies whether the Data conversion data is managed: using the DMA (oneshot or circular), or stored in the DR register or transferred to DFSDM register. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. This parameter can be a value of **ADC Conversion Data Management**. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- **`uint32_t ADC_InitTypeDef::Overrun`**

Select the behavior in case of overrun: data overwritten or preserved (default). This parameter applies to ADC group regular only. This parameter can be a value of **ADC overrun**. Note: In case of overrun set to data preserved and usage with programming model with interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved in function **HAL_ADC_ConvCpltCallback()**, placed in user program code (called before end of conversion flags clear). Note: Error reporting with respect to the conversion mode:

- Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read all the converted data, this is not considered as an erroneous case.
- Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register).

- **`uint32_t ADC_InitTypeDef::LeftBitShift`**

Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of **ADC Extended Oversampling left Shift**

- **`FunctionalState ADC_InitTypeDef::BoostMode`**

Configures the Boost mode control. When selecting an analog ADC clock frequency bigger than 20MHz, it is mandatory to first enable the BOOST Mode. This parameter can be set to ENABLE or DISABLE.

- **`FunctionalState ADC_InitTypeDef::OversamplingMode`**

Specify whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing on ADC groups regular and injected

- **`ADC_OversamplingTypeDef ADC_InitTypeDef::Oversampling`**

Specify the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling is already enabled.

5.1.3 ADC_AnalogWDGConfTypeDef

Data Fields

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

Field Documentation

- *uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber*

Select which ADC analog watchdog is monitoring the selected channel. For Analog Watchdog 1: Only 1 channel can be monitored (or overall group of channels by setting parameter 'WatchdogMode') For Analog Watchdog 2 and 3: Several channels can be monitored (by successive calls of 'HAL_ADC_AnalogWDGConfig()' for each channel) This parameter can be a value of **ADC Extended Analog Watchdog Selection**.

- *uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode*

Configure the ADC analog watchdog mode: single/all/none channels. For Analog Watchdog 1: Configure the ADC analog watchdog mode: single channel/all channels, ADC groups regular and/or injected. For Analog Watchdog 2 and 3: There is no configuration for all channels as AWD1. Set value 'ADC_ANALOGWATCHDOG_NONE' to reset channels group programmed with parameter 'Channel', set any other value to program the channel(s) to be monitored. This parameter can be a value of **ADC Extended Analog Watchdog Mode**.

- *uint32_t ADC_AnalogWDGConfTypeDef::Channel*

Select which ADC channel to monitor by analog watchdog. For Analog Watchdog 1: this parameter has an effect only if parameter 'WatchdogMode' is configured on single channel (only 1 channel can be monitored). For Analog Watchdog 2 and 3: Several channels can be monitored. To use this feature, call successively the function **HAL_ADC_AnalogWDGConfig()** for each channel to be added (or removed with value 'ADC_ANALOGWATCHDOG_NONE'). This parameter can be a value of **ADC Channels**.

- *FunctionalState ADC_AnalogWDGConfTypeDef::ITMode*

Specify whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE

- *uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold*

Configure the ADC analog watchdog High threshold value. Depending of ADC resolution selected (16, 14, 12, 10 or 8 bits), this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF, 0x3FFF, 0xFFFF, 0x3FF or 0xFF respectively.

- *uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold*

Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (16, 14, 12, or 8 bits), this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF, 0x3FFF, 0xFFFF, 0x3FF or 0xFF respectively.

5.2 ADC Firmware driver API description

5.2.1 ADC specific features

- 16-bit, 14-bit, 12-bit, 10-bit or 8-bit configurable resolution.
- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (channel wise)
- External trigger (timer or EXTI) with configurable polarity

- DMA request generation for transfer of conversions data of regular group.
- Configurable delay between conversions in Dual interleaved mode.
- ADC channels selectable single/differential input.
- ADC offset on regular groups.
- ADC calibration
- ADC conversion of regular group.
- ADC supply requirements: 1.62 V to 3.6 V.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

5.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
 - As prerequisite, ADC clock must be configured at RCC top level.
 - Two clock settings are mandatory:
 - ADC clock (core clock, also possibly conversion clock).
 - ADC clock (conversions clock). Two possible clock sources: synchronous clock derived from AHB clock or asynchronous clock derived from system clock, the PLL2 or the PLL3 running up to 400MHz.
 - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
 - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory) RCC_ADCCLKSOURCE_PLL2 enable: (optional: if asynchronous clock selected)
 - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
 - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
 - PeriphClkInit.AdccClockSelection = RCC_ADCCLKSOURCE_PLL2;
 - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
 - ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
 - 2. ADC pins configuration
 - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
 - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
 - 3. Optionally, in case of usage of ADC with interruptions:
 - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
 - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
 - 4. Optionally, in case of usage of DMA:
 - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
 - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)
 - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, group regular, channels parameters

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

Execution of ADC conversions

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

Note:

Callback functions must be implemented in user program:

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

Deinitialization of ADC

1. Disable the ADC interface
 - ADC clock can be hard reset and disabled at RCC top level.
 - Hard reset of ADC peripherals using macro `__HAL_RCC_ADCx_FORCE_RESET()`, `__HAL_RCC_ADCx_RELEASE_RESET()`.
 - ADC clock disable using the equivalent macro/functions as configuration step.
 - Example: Into `HAL_ADC_MspDelinit()` (recommended code location) or with other device clock parameters configuration:
 - `__HAL_RCC_ADC_CLK_DISABLE();` (if not used anymore) `RCC_ADCCLKSOURCE_CLKP` restore: (optional)
 - `RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;`
 - `PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;`
 - `PeriphClkInit.AdclkClockSelection = RCC_ADCCLKSOURCE_CLKP;`
 - `HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);`
2. ADC pins configuration
 - Disable the clock for the ADC GPIOs using macro `__HAL_RCC_GPIOx_CLK_DISABLE()`
3. Optionally, in case of usage of ADC with interruptions:
 - Disable the NVIC for ADC using function `HAL_NVIC_EnableIRQ(ADCx_IRQn)`
4. Optionally, in case of usage of DMA:
 - Deinitialize the DMA using function `HAL_DMA_Init()`.
 - Disable the NVIC for DMA using function `HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)`

5.2.3 Initialization and deinitialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- Deinitialize the ADC.

This section contains the following APIs:

- [*HAL_ADC_Init*](#)
- [*HAL_ADC_DelInit*](#)
- [*HAL_ADC_MspInit*](#)
- [*HAL_ADC_MspDelInit*](#)

5.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion of regular group.
- Stop conversion of regular group.
- Poll for conversion complete on regular group.
- Poll for conversion event.
- Get result of regular channel conversion.
- Start conversion of regular group and enable interruptions.
- Stop conversion of regular group and disable interruptions.
- Handle ADC interrupt request
- Start conversion of regular group and enable DMA transfer.
- Stop conversion of regular group and disable ADC DMA transfer.

This section contains the following APIs:

- [*HAL_ADC_Start*](#)
- [*HAL_ADC_Stop*](#)
- [*HAL_ADC_PollForConversion*](#)
- [*HAL_ADC_PollForEvent*](#)
- [*HAL_ADC_Start_IT*](#)
- [*HAL_ADC_Stop_IT*](#)
- [*HAL_ADC_Start_DMA*](#)
- [*HAL_ADC_Stop_DMA*](#)
- [*HAL_ADC_GetValue*](#)
- [*HAL_ADC_IRQHandler*](#)
- [*HAL_ADC_ConvCpltCallback*](#)
- [*HAL_ADC_ConvHalfCpltCallback*](#)
- [*HAL_ADC_LevelOutOfWindowCallback*](#)
- [*HAL_ADC_ErrorCallback*](#)

5.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- [*HAL_ADC_ConfigChannel*](#)
- [*HAL_ADC_AnalogWDGConfig*](#)

5.2.6 Peripheral state and errors functions

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

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:

- [*HAL_ADC_GetState*](#)
- [*HAL_ADC_GetError*](#)

5.2.7 Detailed description of functions

HAL_ADC_Init

Function name

HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- As prerequisite, ADC clock must be configured at RCC top level depending on possible clock sources: PLL2/PLL3 clocks or AHB clock.
- Possibility to update parameters on the fly: this function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DelInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned by ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef".
- This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".
- Parameters related to common ADC registers (ADC clock mode) are set only if all ADCs are disabled. If this is not the case, these common parameters setting are bypassed without error reporting: it can be the intended behaviour in case of update of a parameter of ADC_InitTypeDef on the fly, without disabling the other ADCs.

HAL_ADC_DelInit

Function name

HAL_StatusTypeDef HAL_ADC_DelInit (ADC_HandleTypeDef * hadc)

Function description

Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Keep in mind that all ADCs use the same clock: disabling the clock will reset all ADCs.
- By default, HAL_ADC_Delnit() sets DEEPPWD: this saves more power by reducing the leakage currents and is particularly interesting before entering STOP 1 or STOP 2 modes.

HAL_ADC_MspInit

Function name

```
void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)
```

Function description

Initialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_MspDelnit

Function name

```
void HAL_ADC_MspDelnit (ADC_HandleTypeDef * hadc)
```

Function description

Deinitialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- All ADCs use the same clock: disabling the clock will reset all ADCs.

HAL_ADC_Start

Function name

```
HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)
```

Function description

Enable ADC, starts conversion of regular group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled(when multimode feature is available): if ADC is Slave, ADC is enabled but conversion is not started, if ADC is master, ADC is enabled and multimode conversion is started.

HAL_ADC_Stop

Function name

HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

HAL_ADC_PollForConversion

Function name

HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for regular group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- Depending on hadc->Init.EOCSelection, EOS or EOC is checked and cleared depending on AUTDLY bit status.
- HAL_ADC_PollForConversion() returns HAL_ERROR if EOC is polled in a DMA-managed conversions configuration: indeed, EOC is immediately reset by the DMA reading the DR register when the converted data is available. Therefore, EOC is set for a too short period to be reliably polled.

HAL_ADC_PollForEvent

Function name

HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)

Function description

Poll for ADC event.

Parameters

- **hadc:** ADC handle
- **EventType:** the ADC event type. This parameter can be one of the following values:
 - ADC_EOSMP_EVENT ADC End of Sampling event
 - ADC_AWD1_EVENT ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 devices)

- ADC_AWD2_EVENT ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 families)
- ADC_AWD3_EVENT ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 families)
- ADC_OVR_EVENT ADC Overrun event
- ADC_JQOVF_EVENT ADC Injected context queue overflow event
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- The relevant flag is cleared if found to be set, except for ADC_FLAG_OVR. Indeed, the latter is reset only if hadc->Init.Overrun field is set to ADC_OVR_DATA_OVERWRITTEN. Otherwise, DR may be potentially overwritten by a new converted data as soon as OVR is cleared. To reset OVR flag once the preserved data is retrieved, the user can resort to macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_OVR);

HAL_ADC_Start_IT

Function name

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group with interruption.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

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

HAL_ADC_Stop_IT

Function name

HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interruption of end-of-conversion, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADC_Start_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start conversion of regular group and transfer result through DMA.

Parameters

- **hadc:** ADC handle
- **pData:** Destination Buffer address.
- **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes)

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function: overrun (if applicable), DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_DMA() is designed for single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStart_DMA() function must be used.

HAL_ADC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Stop_DMA() function is dedicated to single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStop_DMA() API must be used.

HAL_ADC_GetValue

Function name

uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)

Function description

Get ADC regular group conversion result.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** group regular conversion data

Notes

- Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion).
- This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC. If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

HAL_ADC_IRQHandler

Function name

`void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)`

Function description

Handle ADC interrupt request.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvCpltCallback

Function name

`void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)`

Function description

Conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvHalfCpltCallback

Function name

`void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)`

Function description

Conversion DMA half-transfer callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_LevelOutOfWindowCallback

Function name

```
void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)
```

Function description

Analog watchdog 1 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ErrorCallback

Function name

```
void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)
```

Function description

ADC error callback in non-blocking mode (ADC conversion with interruption or transfer by DMA).

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- In case of error due to overrun when using ADC with DMA transfer (HAL ADC handle parameter "ErrorCode" to state "HAL_ADC_ERROR_OVR"): Reinitialize the DMA using function "HAL_ADC_Stop_DMA()". If needed, restart a new ADC conversion using function "HAL_ADC_Start_DMA()" (this function is also clearing overrun flag)

HAL_ADC_ConfigChannel

Function name

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

Function description

Configure a channel to be assigned to ADC group regular.

Parameters

- **hadc:** ADC handle
- **sConfig:** Structure of ADC channel assigned to ADC group regular.

Return values

- **HAL:** status

Notes

- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Possibility to update parameters on the fly: This function initializes channel into ADC group regular, following calls to this function can be used to reconfigure some parameters of structure

"ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name

```
HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc,  
ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)
```

Function description

Configure the analog watchdog.

Parameters

- **hadc:** ADC handle
- **AnalogWDGConfig:** Structure of ADC analog watchdog configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes the selected analog watchdog, successive calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef".
- Analog watchdog thresholds can be modified while ADC conversion is on going. In this case, some constraints must be taken into account: the programmed threshold values are effective from the next ADC EOC (end of unitary conversion). Considering that registers write delay may happen due to bus activity, this might cause an uncertainty on the effective timing of the new programmed threshold values.

HAL_ADC_GetState

Function name

```
uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)
```

Function description

Return the ADC handle state.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** handle state (bitfield on 32 bits)

Notes

- ADC state machine is managed by bitfields, ADC status must be compared with states bits. For example: "if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_REG_BUSY)) " " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_AWD1)) "

HAL_ADC_GetError

Function name

```
uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)
```

Function description

Return the ADC error code.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** error code (bitfield on 32 bits)

ADC_ConversionStop**Function name**

HAL_StatusTypeDef ADC_ConversionStop (ADC_HandleTypeDef * hadc, uint32_t ConversionGroup)

Function description

Stop ADC conversion.

Parameters

- **hadc:** ADC handle
- **ConversionGroup:** ADC group regular and/or injected. This parameter can be one of the following values:
 - ADC_REGULAR_GROUP ADC regular conversion type.
 - ADC_INJECTED_GROUP ADC injected conversion type.
 - ADC_REGULAR_INJECTED_GROUP ADC regular and injected conversion type.

Return values

- **HAL:** status.

ADC_Enable**Function name**

HAL_StatusTypeDef ADC_Enable (ADC_HandleTypeDef * hadc)

Function description

Enable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC must be disabled and voltage regulator must be enabled (done into HAL_ADC_Init()).

ADC_Disable**Function name**

HAL_StatusTypeDef ADC_Disable (ADC_HandleTypeDef * hadc)

Function description

Disable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC conversions must be stopped.

ADC_DMAConvCplt

Function name

```
void ADC_DMAConvCplt (DMA_HandleTypeDef * hdma)
```

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAHalfConvCplt

Function name

```
void ADC_DMAHalfConvCplt (DMA_HandleTypeDef * hdma)
```

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAError

Function name

```
void ADC_DMAError (DMA_HandleTypeDef * hdma)
```

Function description

DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

5.3 ADC Firmware driver defines

5.3.1 ADC

ADC Channels

ADC_CHANNEL_0

ADC_CHANNEL_1

ADC_CHANNEL_2

ADC_CHANNEL_3

ADC_CHANNEL_4

ADC_CHANNEL_5

ADC_CHANNEL_6

ADC_CHANNEL_7

ADC_CHANNEL_8

ADC_CHANNEL_9

ADC_CHANNEL_10

ADC_CHANNEL_11

ADC_CHANNEL_12

ADC_CHANNEL_13

ADC_CHANNEL_14

ADC_CHANNEL_15

ADC_CHANNEL_16

ADC_CHANNEL_17

ADC_CHANNEL_18

ADC_CHANNEL_19

ADC_CHANNEL_VBAT_DIV4

ADC_CHANNEL_TEMPSENSOR

ADC_CHANNEL_VREFINT

ADC_CHANNEL_DAC1CH1_ADC2

ADC internal channel connected to DAC1 channel 1, channel specific to ADC2

ADC_CHANNEL_DAC1CH2_ADC2

ADC internal channel connected to DAC1 channel 2, channel specific to ADC2

ADC clock source and clock prescaler

ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock derived from AHB clock not divided

ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock derived from AHB clock divided by 2

ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock derived from AHB clock divided by 4

ADC_CLOCKPRESCALER_PCLK_DIV1

Obsolete naming, kept for compatibility with some other devices

ADC_CLOCKPRESCALER_PCLK_DIV2

Obsolete naming, kept for compatibility with some other devices

ADC_CLOCKPRESCALER_PCLK_DIV4

Obsolete naming, kept for compatibility with some other devices

ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock not divided

ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock divided by 2

ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock divided by 4

ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock divided by 6

ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock divided by 8

ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock divided by 10

ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock divided by 12

ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock divided by 16

ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock divided by 32

ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock divided by 64

ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock divided by 128

ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock divided by 256

ADC Conversion Data Management**ADC_CONVERSIONDATA_DR**

Regular Conversion data stored in DR register only

ADC_CONVERSIONDATA_DFSDM

DFSDM mode selected

ADC_CONVERSIONDATA_DMA_ONESHOT

DMA one shot mode selected

ADC_CONVERSIONDATA_DMA_CIRCULAR

DMA circular mode selected

ADC sequencer end of unitary conversion or sequence conversions

ADC_EOC_SINGLE_CONV

End of unitary conversion flag

ADC_EOC_SEQ_CONV

End of sequence conversions flag

ADC Error Code

HAL_ADC_ERROR_NONE

No error

HAL_ADC_ERROR_INTERNAL

ADC IP internal error (problem of clocking, enable/disable, erroneous state, ...)

HAL_ADC_ERROR_OVR

Overrun error

HAL_ADC_ERROR_DMA

DMA transfer error

HAL_ADC_ERROR_JQOVF

Injected context queue overflow error

ADC Exported Macros

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be one of the following values:
 - ADC_IT_RDY ADC Ready (ADRDY) interrupt source
 - ADC_IT_EOSMP ADC End of Sampling interrupt source
 - ADC_IT_EOC ADC End of Regular Conversion interrupt source
 - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
 - ADC_IT_OVR ADC overrun interrupt source
 - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
 - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
 - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)

- ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
- ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
- ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source

Return value:

- State: of interruption (SET or RESET)

[__HAL_ADC_ENABLE_IT](#)**Description:**

- Enable an ADC interrupt.

Parameters:

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

Return value:

- None

[__HAL_ADC_DISABLE_IT](#)**Description:**

- Disable an ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt to disable
 - ADC_IT_RDY ADC Ready (ADRDY) interrupt source
 - ADC_IT_EOSMP ADC End of Sampling interrupt source
 - ADC_IT_EOC ADC End of Regular Conversion interrupt source
 - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
 - ADC_IT_OVR ADC overrun interrupt source
 - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
 - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
 - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
 - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
 - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
 - ADC_IT_JQOVF ADC Injected Context Queue Overflow interrupt source

Return value:

- None

[__HAL_ADC_GET_FLAG](#)**Description:**

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

Parameters:

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

Return value:

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

[`__HAL_ADC_CLEAR_FLAG`](#)**Description:**

- Clear a specified ADC flag.

Parameters:

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

Return value:

- None

Notes:

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

ADC group injected trigger edge (when external trigger is selected)[`ADC_EXTERNALTRIGINJECCONV_EDGE_NONE`](#)

Injected conversions hardware trigger detection disabled

[`ADC_EXTERNALTRIGINJECCONV_EDGE_RISING`](#)

Injected conversions hardware trigger detection on the rising edge

ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING

Injected conversions hardware trigger detection on the falling edge

ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING

Injected conversions hardware trigger detection on both the rising and falling edges

ADC overrun**ADC_OVR_DATA_PRESERVED**

Data preserved in case of overrun

ADC_OVR_DATA_OVERWRITTEN

Data overwritten in case of overrun

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

Regular conversions hardware trigger detection disabled

ADC_EXTERNALTRIGCONVEDGE_RISING

Regular conversions hardware trigger detection on the rising edge

ADC_EXTERNALTRIGCONVEDGE_FALLING

Regular conversions hardware trigger detection on the falling edge

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

Regular conversions hardware trigger detection on both the rising and falling edges

ADC group regular trigger source**ADC_EXTERNALTRIG_T1_CC1****ADC_EXTERNALTRIG_T1_CC2****ADC_EXTERNALTRIG_T1_CC3****ADC_EXTERNALTRIG_T2_CC2****ADC_EXTERNALTRIG_T3_TRGO****ADC_EXTERNALTRIG_T4_CC4****ADC_EXTERNALTRIG_EXT_IT11****ADC_EXTERNALTRIG_T8_TRGO****ADC_EXTERNALTRIG_T8_TRGO2****ADC_EXTERNALTRIG_T1_TRGO****ADC_EXTERNALTRIG_T1_TRGO2****ADC_EXTERNALTRIG_T2_TRGO****ADC_EXTERNALTRIG_T4_TRGO****ADC_EXTERNALTRIG_T6_TRGO**

ADC_EXTERNALTRIG_T15_TRGO
ADC_EXTERNALTRIG_T3_CC4
ADC_EXTERNALTRIG_HR1_ADCTRG1
ADC_EXTERNALTRIG_HR1_ADCTRG3
ADC_EXTERNALTRIG_LPTIM1_OUT
ADC_EXTERNALTRIG_LPTIM2_OUT
ADC_EXTERNALTRIG_LPTIM3_OUT
ADC_SOFTWARE_START

ADC group regular sequencer rank

ADC_REGULAR_RANK_1
ADC regular conversion rank 1
ADC_REGULAR_RANK_2
ADC regular conversion rank 2
ADC_REGULAR_RANK_3
ADC regular conversion rank 3
ADC_REGULAR_RANK_4
ADC regular conversion rank 4
ADC_REGULAR_RANK_5
ADC regular conversion rank 5
ADC_REGULAR_RANK_6
ADC regular conversion rank 6
ADC_REGULAR_RANK_7
ADC regular conversion rank 7
ADC_REGULAR_RANK_8
ADC regular conversion rank 8
ADC_REGULAR_RANK_9
ADC regular conversion rank 9
ADC_REGULAR_RANK_10
ADC regular conversion rank 10
ADC_REGULAR_RANK_11
ADC regular conversion rank 11
ADC_REGULAR_RANK_12
ADC regular conversion rank 12
ADC_REGULAR_RANK_13
ADC regular conversion rank 13

ADC_REGULAR_RANK_14

ADC regular conversion rank 14

ADC_REGULAR_RANK_15

ADC regular conversion rank 15

ADC_REGULAR_RANK_16

ADC regular conversion rank 16

ADC Resolution**ADC_RESOLUTION_16B**

ADC 16-bit resolution

ADC_RESOLUTION_14B

ADC 14-bit resolution

ADC_RESOLUTION_12B

ADC 12-bit resolution

ADC_RESOLUTION_10B

ADC 10-bit resolution

ADC_RESOLUTION_8B

ADC 8-bit resolution

ADC Sampling Times**ADC_SAMPLETIME_1CYCLE_5**

Sampling time 1.5 ADC clock cycle

ADC_SAMPLETIME_2CYCLES_5

Sampling time 2.5 ADC clock cycles

ADC_SAMPLETIME_8CYCLES_5

Sampling time 8.5 ADC clock cycles

ADC_SAMPLETIME_16CYCLES_5

Sampling time 16.5 ADC clock cycles

ADC_SAMPLETIME_32CYCLES_5

Sampling time 32.5 ADC clock cycles

ADC_SAMPLETIME_64CYCLES_5

Sampling time 64.5 ADC clock cycles

ADC_SAMPLETIME_387CYCLES_5

Sampling time 387.5 ADC clock cycles

ADC_SAMPLETIME_810CYCLES_5

Sampling time 810.5 ADC clock cycles

ADC sequencer scan mode**ADC_SCAN_DISABLE**

Scan mode disabled

ADC_SCAN_ENABLE

Scan mode enabled

ADC States

HAL_ADC_STATE_RESET

Notes:

- ADC state machine is managed by bitfields, state must be compared with bit by bit. For example: " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_REG_BUSY)) " " if (HAL_IS_BIT_SET(HAL_ADC_GetState(hadc1), HAL_ADC_STATE_AWD1)) " ADC not yet initialized or disabled

HAL_ADC_STATE_READY

ADC peripheral ready for use

HAL_ADC_STATE_BUSY_INTERNAL

ADC is busy due to an internal process (initialization, calibration)

HAL_ADC_STATE_TIMEOUT

TimeOut occurrence

HAL_ADC_STATE_ERROR_INTERNAL

Internal error occurrence

HAL_ADC_STATE_ERROR_CONFIG

Configuration error occurrence

HAL_ADC_STATE_ERROR_DMA

DMA error occurrence

HAL_ADC_STATE_REG_BUSY

A conversion on ADC group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_REG_EOC

Conversion data available on group regular

HAL_ADC_STATE_REG_OVR

Overrun occurrence

HAL_ADC_STATE_REG_EOSMP

Not available on this STM32 serie: End Of Sampling flag raised

HAL_ADC_STATE_INJ_BUSY

A conversion on ADC group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_INJ_EOC

Conversion data available on group injected

HAL_ADC_STATE_INJ_JQOVF

Injected queue overflow occurrence

HAL_ADC_STATE_AWD1

Out-of-window occurrence of ADC analog watchdog 1

HAL_ADC_STATE_AWD2

Out-of-window occurrence of ADC analog watchdog 2

HAL_ADC_STATE_AWD3

Out-of-window occurrence of ADC analog watchdog 3

HAL_ADC_STATE_MULTIMODE_SLAVE

ADC in multimode slave state, controlled by another ADC master (when feature available)

6 HAL ADC Extension Driver

6.1 ADCEx Firmware driver registers structures

6.1.1 ADC_InjectionConfigTypeDef

Data Fields

- *uint32_t ContextQueue*
- *uint32_t ChannelCount*

Field Documentation

- *uint32_t ADC_InjectionConfigTypeDef::ContextQueue*

Injected channel configuration context: build-up over each **HAL_ADCEx_InjectedConfigChannel()** call to finally initialize JSQR register at **HAL_ADCEx_InjectedConfigChannel()** last call

- *uint32_t ADC_InjectionConfigTypeDef::ChannelCount*

Number of channels in the injected sequence

6.1.2 ADC_HandleTypeDefDef

Data Fields

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

Field Documentation

- *ADC_TypeDef* ADC_HandleTypeDefDef::Instance*

Register base address

- *ADC_InitTypeDef ADC_HandleTypeDefDef::Init*

ADC initialization parameters and regular conversions setting

- *DMA_HandleTypeDef* ADC_HandleTypeDefDef::DMA_Handle*

Pointer DMA Handler

- *HAL_LockTypeDef ADC_HandleTypeDefDef::Lock*

ADC locking object

- *__IO uint32_t ADC_HandleTypeDefDef::State*

ADC communication state (bit-map of ADC states)

- *__IO uint32_t ADC_HandleTypeDefDef::ErrorCode*

ADC Error code

- *ADC_InjectionConfigTypeDef ADC_HandleTypeDefDef::InjectionConfig*

ADC injected channel configuration build-up structure

6.1.3 ADC_InjOversamplingTypeDef

Data Fields

- `uint32_t Ratio`
- `uint32_t RightBitShift`

Field Documentation

- `uint32_t ADC_InjOversamplingTypeDef::Ratio`

Configures the oversampling ratio. This parameter can be a value between 0 to 1023

- `uint32_t ADC_InjOversamplingTypeDef::RightBitShift`

Configures the division coefficient for the Oversampler. This parameter can be a value of **ADC Extended Oversampling Right Shift**

6.1.4 ADC_ChannelConfTypeDef

Data Fields

- `uint32_t Channel`
- `uint32_t Rank`
- `uint32_t SamplingTime`
- `uint32_t SingleDiff`
- `uint32_t OffsetNumber`
- `uint32_t Offset`
- `FunctionalState OffsetRightShift`
- `FunctionalState OffsetSignedSaturation`

Field Documentation

- `uint32_t ADC_ChannelConfTypeDef::Channel`

Specify the channel to configure into ADC regular group. This parameter can be a value of **ADC Channels**

Note: Depending on devices and ADC instances, some channels may not be available on device package

pins. Refer to device DataSheet for channels availability.

- `uint32_t ADC_ChannelConfTypeDef::Rank`

Specify the rank in the regular group sequencer. This parameter can be a value of **ADC group regular sequencer rank** Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- `uint32_t ADC_ChannelConfTypeDef::SamplingTime`

Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time This parameter can be a value of **ADC Sampling Times**

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

- `uint32_t ADC_ChannelConfTypeDef::SingleDiff`

Select single-ended or differential input. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of **ADC Extended**

Single-ended/Differential input mode Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is

bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)

- **`uint32_t ADC_ChannelConfTypeDef::OffsetNumber`**

Select the offset number This parameter can be a value of **ADC Extended Offset Number** Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.

- **`uint32_t ADC_ChannelConfTypeDef::Offset`**

Define the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (16, 14, 12, 10 or 8 bits), this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF, 0x3FFF, 0xFFF, 0x3FF or 0xFF respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

- **`FunctionalState ADC_ChannelConfTypeDef::OffsetRightShift`**

Define the Right-shift data after Offset correction. This parameter is applied only for 16-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.

- **`FunctionalState ADC_ChannelConfTypeDef::OffsetSignedSaturation`**

Specify whether the Signed saturation feature is used or not. This parameter is applied only for 16-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.

6.1.5 ADC_InjectionConfTypeDef

Data Fields

- **`uint32_t InjectedChannel`**
- **`uint32_t InjectedRank`**
- **`uint32_t InjectedSamplingTime`**
- **`uint32_t InjectedSingleDiff`**
- **`uint32_t InjectedOffsetNumber`**
- **`uint32_t InjectedOffset`**
- **`uint32_t InjectedOffsetRightShift`**
- **`FunctionalState InjectedOffsetSignedSaturation`**
- **`uint32_t InjectedLeftBitShift`**
- **`uint32_t InjectedNbrOfConversion`**
- **`FunctionalState InjectedDiscontinuousConvMode`**
- **`FunctionalState AutoInjectedConv`**
- **`FunctionalState QueueInjectedContext`**
- **`uint32_t ExternalTrigInjecConv`**
- **`uint32_t ExternalTrigInjecConvEdge`**
- **`FunctionalState InjecOversamplingMode`**
- **`ADC_InjOversamplingTypeDef InjecOversampling`**

Field Documentation

- **`uint32_t ADC_InjectionConfTypeDef::InjectedChannel`**

Specifies the channel to configure into ADC group injected. This parameter can be a value of **ADC Channels** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.

- **`uint32_t ADC_InjectionConfTypeDef::InjectedRank`**

Specifies the rank in the ADC group injected sequencer. This parameter must be a value of **ADC Extended Injected Channel Rank** . Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- **`uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime`**

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

- **`uint32_t ADC_InjectionConfTypeDef::InjectedSingleDiff`**

Selection of single-ended or differential input. In differential mode: Differential measurement is between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of **ADC Extended Single-ended/Differential input mode**. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)
- **`uint32_t ADC_InjectionConfTypeDef::InjectedOffsetNumber`**

Selects the offset number. This parameter can be a value of **ADC Extended Offset Number**. Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- **`uint32_t ADC_InjectionConfTypeDef::InjectedOffset`**

Defines the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (16, 14, 12, 10 or 8bits), this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF, 0x3FFF, 0xFFF, 0x3FF or 0xFF respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- **`uint32_t ADC_InjectionConfTypeDef::InjectedOffsetRightShift`**

Defines the Right-shift data after Offset correction. This parameter is applied only for 16-bit or 8-bit resolution. This parameter must be a value of **ADC Extended Oversampling Right Shift**.
- **`FunctionalState ADC_InjectionConfTypeDef::InjectedOffsetSignedSaturation`**

Specifies whether the Signed saturation feature is used or not. This parameter is applied only for 16-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.
- **`uint32_t ADC_InjectionConfTypeDef::InjectedLeftBitShift`**

Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of **ADC Extended Oversampling left Shift**
- **`uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion`**

Specifies the number of ranks that will be converted within the ADC group injected sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- **`FunctionalState ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode`**

Specifies whether the conversions sequence of ADC group injected is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). Note: For injected group, discontinuous mode converts the sequence channel by channel (discontinuous length fixed to 1 rank). Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- **`FunctionalState ADC_InjectionConfTypeDef::AutoInjectedConv`**

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

- ***FunctionalState ADC_InjectionConfTypeDef::QueueInjectedContext***

Specifies whether the context queue feature is enabled. This parameter can be set to ENABLE or DISABLE. If context queue is enabled, injected sequencer&channels configurations are queued on up to 2 contexts. If a new injected context is set when queue is full, error is triggered by interruption and through function '**HAL_ADCEx_InjectedQueueOverflowCallback**'. Caution: This feature request that the sequence is fully configured before injected conversion start. Therefore, configure channels with as many calls to **HAL_ADCEx_InjectedConfigChannel()** as the 'InjectedNbrOfConversion' parameter. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion).

- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***

Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of **ADC Extended External Trigger Source for Injected Group**. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***

Selects the external trigger edge of injected group. This parameter can be a value of **ADC group injected trigger edge (when external trigger is selected)**. If trigger source is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***FunctionalState ADC_InjectionConfTypeDef::InjecOversamplingMode***

Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared)

- ***ADC_InjOversamplingTypeDef ADC_InjectionConfTypeDef::InjecOversampling***

Specifies the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling already enabled. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).

6.1.6 ADC_MultiModeTypeDef

Data Fields

- ***uint32_t Mode***
- ***uint32_t DualModeData***
- ***uint32_t TwoSamplingDelay***

Field Documentation

- ***uint32_t ADC_MultiModeTypeDef::Mode***

Configures the ADC to operate in independent or MultiMode. This parameter can be a value of **ADC Extended multimode dual mode**

- ***uint32_t ADC_MultiModeTypeDef::DualModeData***

Configures the Dual ADC Mode Data Format: This parameter can be a value of **ADC Extended Dual Mode Data Formatting**

- **`uint32_t ADC_MultiModeTypeDef::TwoSamplingDelay`**

Configures the Delay between 2 sampling phases. This parameter can be a value of **ADC Extended delay between 2 sampling phases** Delay range depends on selected resolution: from 1 to 9 clock cycles for 16 bits, from 1 to 9 clock cycles for 14 bits from 1 to 8 clock cycles for 12 bits from 1 to 6 clock cycles for 10 bits from 1 to 6 clock cycles for 8 bits

6.2 ADCEx Firmware driver API description

6.2.1 ADC specific features

Note: Sections "ADC peripheral features" and "How to use this driver" are available in file of generic functions "stm32h7xx_hal_adc.c".

6.2.2 IO operation functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [**HAL_ADCEx_Calibration_Start**](#)
- [**HAL_ADCEx_Calibration_GetValue**](#)
- [**HAL_ADCEx_LinearCalibration_GetValue**](#)
- [**HAL_ADCEx_Calibration_SetValue**](#)
- [**HAL_ADCEx_LinearCalibration_SetValue**](#)
- [**HAL_ADCEx_InjectedStart**](#)
- [**HAL_ADCEx_InjectedStop**](#)
- [**HAL_ADCEx_InjectedPollForConversion**](#)
- [**HAL_ADCEx_InjectedStart_IT**](#)
- [**HAL_ADCEx_InjectedStop_IT**](#)
- [**HAL_ADCEx_MultiModeStart_DMA**](#)
- [**HAL_ADCEx_MultiModeStop_DMA**](#)
- [**HAL_ADCEx_MultiModeGetValue**](#)
- [**HAL_ADCEx_InjectedGetValue**](#)
- [**HAL_ADCEx_InjectedConvCpltCallback**](#)
- [**HAL_ADCEx_InjectedQueueOverflowCallback**](#)
- [**HAL_ADCEx_LevelOutOfWindow2Callback**](#)
- [**HAL_ADCEx_LevelOutOfWindow3Callback**](#)

- [*HAL_ADCEx_EndOfSamplingCallback*](#)
- [*HAL_ADCEx-RegularStop*](#)
- [*HAL_ADCEx-RegularStop_IT*](#)
- [*HAL_ADCEx-RegularStop_DMA*](#)
- [*HAL_ADCEx-RegularMultiModeStop_DMA*](#)

6.2.3 Peripheral Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [*HAL_ADCEx_InjectedConfigChannel*](#)
- [*HAL_ADCEx_MultiModeConfigChannel*](#)
- [*HAL_ADCEx_EnableInjectedQueue*](#)
- [*HAL_ADCEx_DisableInjectedQueue*](#)
- [*HAL_ADCEx_DisableVoltageRegulator*](#)
- [*HAL_ADCEx_EnterADCDeepPowerDownMode*](#)

6.2.4 Detailed description of functions

`HAL_ADCEx_Calibration_Start`

Function name

`HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t CalibrationMode, uint32_t SingleDiff)`

Function description

Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before `HAL_ADC_Start()` or after `HAL_ADC_Stop()`).

Parameters

- **hadc:** ADC handle
- **CalibrationMode:** Selection of Calibration Mode This parameter can be one of the following values:
 - `ADC_CALIB_OFFSET`: ADC calibration in offset mode
 - `ADC_CALIB_OFFSET_LINEARITY`: ADC calibration in Linear offset mode
- **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values:
 - `ADC_SINGLE_ENDED`: Channel in mode input single ended
 - `ADC_DIFFERENTIAL_ENDED`: Channel in mode input differential ended

Return values

- **HAL:** status

`HAL_ADCEx_Calibration_GetValue`

Function name

`uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)`

Function description

Get the calibration factor from automatic conversion result.

Parameters

- **hadc:** ADC handle.
- **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values:
 - ADC_SINGLE_ENDED Channel in mode input single ended
 - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **HAL:** state

HAL_ADCEx_LinearCalibration_GetValue

Function name

```
HAL_StatusTypeDef HAL_ADCEx_LinearCalibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t *  
LinearCalib_Buffer)
```

Function description

Get the calibration factor from automatic conversion result.

Parameters

- **hadc:** ADC handle
- **LinearCalib_Buffer:** Linear calibration factor

Return values

- **HAL:** state

HAL_ADCEx_Calibration_SetValue

Function name

```
HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff,  
uint32_t CalibrationFactor)
```

Function description

Set the calibration factor to overwrite automatic conversion result, ADC must be enabled and no conversion on going.

Parameters

- **hadc:** ADC handle.
- **SingleDiff:** Selection of single-ended or differential input. This parameter can be one of the following values:
 - ADC_SINGLE_ENDED Channel in mode input single ended
 - ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
- **CalibrationFactor:** Calibration factor (coded on 7 bits maximum)

Return values

- **HAL:** state

HAL_ADCEx_LinearCalibration_SetValue

Function name

```
HAL_StatusTypeDef HAL_ADCEx_LinearCalibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t *  
LinearCalib_Buffer)
```

Function description

Set the linear calibration factor.

Parameters

- **hadc:** ADC handle
- **LinearCalib_Buffer:** Linear calibration factor

Return values

- **HAL:** state

HAL_ADCEx_InjectedStart

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

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

HAL_ADCEx_InjectedStop

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels and disable ADC peripheral if no regular conversion is on going.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- In case of multimode enabled (when multimode feature is available), HAL_ADCEx_InjectedStop() must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

HAL_ADCEx_InjectedPollForConversion

Function name

```
HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)
```

Function description

Wait for injected group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- Depending on hadc->Init.EOCSelection, JEOS or JEOC is checked and cleared depending on AUTDLY bit status.

HAL_ADCEx_InjectedStart_IT

Function name

```
HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)
```

Function description

Enable ADC, start conversion of injected group with interruption.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function according to initialization setting : JEOC (end of conversion) or JEOS (end of sequence)
- Case of multimode enabled (when multimode feature is enabled): HAL_ADCEx_InjectedStart_IT() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop_IT

Function name

```
HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)
```

Function description

Stop conversion of injected channels, disable interruption of end-of-conversion, disable ADC peripheral if no regular conversion is on going.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- Case of multimode enabled (when multimode feature is available): HAL_ADCEx_InjectedStop_IT() API must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).
- In case of auto-injection mode, HAL_ADC_Stop() must be used.

HAL_ADCEx_MultiModeStart_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start MultiMode conversion and transfer regular results through DMA.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)
- **pData:** Destination Buffer address.
- **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes).

Return values

- **HAL:** status

Notes

- Multimode must have been previously configured using HAL_ADCEx_MultiModeConfigChannel() function. Interruptions enabled in this function: overrun, DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- State field of Slave ADC handle is not updated in this configuration: user should not rely on it for information related to Slave regular conversions.

HAL_ADCEx_MultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. MultiMode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable Multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DelInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADC_Stop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_MultiModeGetValue

Function name

```
uint32_t HAL_ADCEx_MultiModeGetValue (ADC_HandleTypeDef * hadc)
```

Function description

Return the last ADC Master and Slave regular conversions results when in multimode configuration.

Parameters

- **hadc:** ADC handle of ADC Master (handle of ADC Slave must not be used)

Return values

- **The:** converted data values.

HAL_ADCEx_InjectedGetValue

Function name

```
uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef * hadc, uint32_t InjectedRank)
```

Function description

Get ADC injected group conversion result.

Parameters

- **hadc:** ADC handle
- **InjectedRank:** the converted ADC injected rank. This parameter can be one of the following values:
 - ADC_INJECTED_RANK_1 ADC group injected rank 1
 - ADC_INJECTED_RANK_2 ADC group injected rank 2
 - ADC_INJECTED_RANK_3 ADC group injected rank 3
 - ADC_INJECTED_RANK_4 ADC group injected rank 4

Return values

- **ADC:** group injected conversion data

Notes

- Reading register JDRx automatically clears ADC flag JEOC (ADC group injected end of unitary conversion).
- This function does not clear ADC flag JEOS (ADC group injected end of sequence conversion) Occurrence of flag JEOS rising: If sequencer is composed of 1 rank, flag JEOS is equivalent to flag JEOC. If sequencer is composed of several ranks, during the scan sequence flag JEOC only is raised, at the end of the scan sequence both flags JEOC and EOS are raised. Flag JEOS must not be cleared by this function because it would not be compliant with low power features (feature low power auto-wait, not available on all STM32 families). To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADCEx_InjectedPollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_JEOS).

HAL_ADCEx_InjectedConvCpltCallback

Function name

```
void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)
```

Function description

Injected conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_InjectedQueueOverflowCallback

Function name

void HAL_ADCEx_InjectedQueueOverflowCallback (ADC_HandleTypeDef * hadc)

Function description

Injected context queue overflow callback.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- This callback is called if injected context queue is enabled (parameter "QueueInjectedContext" in injected channel configuration) and if a new injected context is set when queue is full (maximum 2 contexts).

HAL_ADCEx_LevelOutOfWindow2Callback

Function name

void HAL_ADCEx_LevelOutOfWindow2Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 2 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_LevelOutOfWindow3Callback

Function name

void HAL_ADCEx_LevelOutOfWindow3Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 3 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_EndOfSamplingCallback

Function name

void HAL_ADCEx_EndOfSamplingCallback (ADC_HandleTypeDef * hadc)

Function description

End Of Sampling callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_RegularStop

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of ADC groups regular and injected, disable interruption of end-of-conversion, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- HAL_ADCEx_RegularStop_DMA() function is dedicated to single-ADC mode only. For multimode (when multimode feature is available), HAL_ADCEx_RegularMultiModeStop_DMA() API must be used.

HAL_ADCEx_RegularMultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularMultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop DMA-based MultiMode ADC conversion, disable ADC DMA transfer, disable ADC peripheral if no injected conversion is on-going.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- MultiMode is kept enabled after this function. MultiMode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable MultiMode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DelInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADCEx-RegularStop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_InjectedConfigChannel

Function name

**HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc,
ADC_InjectionConfTypeDef * sConfigInjected)**

Function description

Configure a channel to be assigned to ADC group injected.

Parameters

- **hadc:** ADC handle
- **sConfigInjected:** Structure of ADC injected group and ADC channel for injected group.

Return values

- **HAL:** status

Notes

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

channels x 2 contexts = 6 calls). Conversion can start once the 1st context is set, that is after the first three HAL_ADCEx_InjectedConfigChannel() calls. The 2nd context can be set on the fly.

HAL_ADCEx_MultiModeConfigChannel

Function name

**HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel (ADC_HandleTypeDef * hadc,
ADC_MultiModeTypeDef * multimode)**

Function description

Enable ADC multimode and configure multimode parameters.

Parameters

- **hadc:** Master ADC handle
- **multimode:** : Structure of ADC multimode configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes multimode parameters, following calls to this function can be used to reconfigure some parameters of structure "ADC_MultiModeTypeDef" on the fly, without resetting the ADCs. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_MultiModeTypeDef".
- To move back configuration from multimode to single mode, ADC must be reset (using function HAL_ADC_Init()).

HAL_ADCEx_EnableInjectedQueue

Function name

HAL_StatusTypeDef HAL_ADCEx_EnableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description

Enable Injected Queue.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

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

HAL_ADCEx_DisableInjectedQueue

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableInjectedQueue (ADC_HandleTypeDef * hadc)

Function description

Disable Injected Queue.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

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

HAL_ADCEx_DisableVoltageRegulator

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableVoltageRegulator (ADC_HandleTypeDef * hadc)

Function description

Disable ADC voltage regulator.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Disabling voltage regulator allows to save power. This operation can be carried out only when ADC is disabled.
- To enable again the voltage regulator, the user is expected to resort to HAL_ADC_Init() API.

HAL_ADCEx_EnterADCDeepPowerDownMode

Function name

HAL_StatusTypeDef HAL_ADCEx_EnterADCDeepPowerDownMode (ADC_HandleTypeDef * hadc)

Function description

Enter ADC deep-power-down mode.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This mode is achieved in setting DEEPPWD bit and allows to save power in reducing leakage currents. It is particularly interesting before entering STOP1 or STOP2 modes.
- Setting DEEPPWD automatically clears ADVREGEN bit and disables the ADC voltage regulator. This means that this API encompasses HAL_ADCEx_DisableVoltageRegulator(). Additionally, the internal calibration is lost.
- To exit the ADC deep-power-down mode, the user is expected to resort to HAL_ADC_Init() API as well as to relaunch a calibration with HAL_ADCEx_Calibration_Start() API or to re-apply a previously saved calibration factor.

6.3 ADCEx Firmware driver defines

6.3.1 ADCEx

ADC Extended Analog Watchdog Mode

ADC_ANALOGWATCHDOG_NONE

No analog watchdog selected

ADC_ANALOGWATCHDOG_SINGLE_REG

Analog watchdog applied to a regular group single channel

ADC_ANALOGWATCHDOG_SINGLE_INJEC

Analog watchdog applied to an injected group single channel

ADC_ANALOGWATCHDOG_SINGLE_REGINJEC

Analog watchdog applied to a regular and injected groups single channel

ADC_ANALOGWATCHDOG_ALL_REG

Analog watchdog applied to regular group all channels

ADC_ANALOGWATCHDOG_ALL_INJEC

Analog watchdog applied to injected group all channels

ADC_ANALOGWATCHDOG_ALL_REGINJEC

Analog watchdog applied to regular and injected groups all channels

ADC Extended Analog Watchdog Selection**ADC_ANALOGWATCHDOG_1**

Analog watchdog 1 selection

ADC_ANALOGWATCHDOG_2

Analog watchdog 2 selection

ADC_ANALOGWATCHDOG_3

Analog watchdog 3 selection

ADC Extended Calibration mode offset mode or linear mode**ADC_CALIB_OFFSET****ADC_CALIB_OFFSET_LINEARITY*****ADC Extended multimode dual mode*****ADC_MODE_INDEPENDENT**

Independent ADC conversions mode

ADC_DUALMODE_REGSIMULT_INJECSIMULT

Combined regular simultaneous + injected simultaneous mode

ADC_DUALMODE_REGSIMULT_ALTERTRIG

Combined regular simultaneous + alternate trigger mode

ADC_DUALMODE_REGINTERL_INJECSIMULT

Combined Interleaved mode + injected simultaneous mode

ADC_DUALMODE_INJECSIMULT

Injected simultaneous mode only

ADC_DUALMODE_REGSIMULT

Regular simultaneous mode only

ADC_DUALMODE_INTERL

Interleaved mode only

ADC_DUALMODE_ALTERTRIG

Alternate trigger mode only

ADC Extended Conversion Group**ADC_REGULAR_GROUP**

ADC regular group selection

ADC_INJECTED_GROUP

ADC injected group selection

ADC_REGULAR_INJECTED_GROUP

ADC regular and injected groups selection

ADC Extended delay between 2 sampling phases**ADC_TWOSAMPLINGDELAY_1CYCLE**

1 ADC clock cycle delay

ADC_TWOSAMPLINGDELAY_2CYCLES

2 ADC clock cycles delay

ADC_TWOSAMPLINGDELAY_3CYCLES

3 ADC clock cycles delay

ADC_TWOSAMPLINGDELAY_4CYCLES

4 ADC clock cycles delay (lower for less than 10-bit resolution)

ADC_TWOSAMPLINGDELAY_5CYCLES

5 ADC clock cycles delay (lower for less than 12-bit resolution)

ADC_TWOSAMPLINGDELAY_6CYCLES

6 ADC clock cycles delay (lower for less than 14-bit resolution)

ADC_TWOSAMPLINGDELAY_7CYCLES

7 ADC clock cycles delay (lower for less than 16-bit resolution)

ADC_TWOSAMPLINGDELAY_8CYCLES

8 ADC clock cycles delay (lower for less than 16-bit resolution)

ADC_TWOSAMPLINGDELAY_9CYCLES

9 ADC clock cycles delay (lower for less than 16-bit resolution)

ADC Extended Dual Mode Data Formatting**ADC_DUALMODEDATAFORMAT_DISABLED**

Dual ADC mode without data packing: ADCx_CDR and ADCx_CDR2 registers not used

ADC_DUALMODEDATAFORMAT_32_10_BITS

Data formatting mode for 32 down to 10-bit resolution

ADC_DUALMODEDATAFORMAT_8_BITS

Data formatting mode for 8-bit resolution

ADC Extended Event Type

ADC_EOSMP_EVENT

ADC End of Sampling event

ADC_AWD1_EVENT

ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 series)

ADC_AWD2_EVENT

ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 series)

ADC_AWD3_EVENT

ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 series)

ADC_OVR_EVENT

ADC overrun event

ADC_JQOVF_EVENT

ADC Injected Context Queue Overflow event

ADC Extended Exported Constants**ADC_AWD_EVENT**

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

ADC Extended Flags Definition**ADC_FLAG_RDY**

ADC Ready (ADRDY) flag

ADC_FLAG_EOSMP

ADC End of Sampling flag

ADC_FLAG_EOC

ADC End of Regular Conversion flag

ADC_FLAG_EOS

ADC End of Regular sequence of Conversions flag

ADC_FLAG_OVR

ADC overrun flag

ADC_FLAG_JEOC

ADC End of Injected Conversion flag

ADC_FLAG_JEOS

ADC End of Injected sequence of Conversions flag

ADC_FLAG_AWD1

ADC Analog watchdog 1 flag (main analog watchdog)

ADC_FLAG_AWD2

ADC Analog watchdog 2 flag (additional analog watchdog)

ADC_FLAG_AWD3

ADC Analog watchdog 3 flag (additional analog watchdog)

ADC_FLAG_JQOVF

ADC Injected Context Queue Overflow flag

ADC_FLAG_AWD

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

ADC_FLAG_ALL

ADC all flags

ADC_FLAG_POSTCONV_ALL

ADC post-conversion all flags

ADC Extended External Trigger Source for Injected Group**ADC_EXTERNALTRIGINJEC_T1_TRGO**

Event 0 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T1_CC4

Event 1 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T2_TRGO

Event 2 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T2_CC1

Event 3 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T3_CC4

Event 4 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T4_TRGO

Event 5 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_EXT_IT15

Event 6 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T8_CC4

Event 7 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T1_TRGO2

Event 8 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T8_TRGO

Event 9 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T8_TRGO2

Event 10 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T3_CC3

Event 11 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T3_TRGO

Event 12 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T3_CC1

Event 13 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T6_TRGO

Event 14 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T15_TRGO

Event 15 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_HR1_ADCTRG2

Event 16 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_HR1_ADCTRG4

Event 17 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_LPTIM1_OUT

Event 18 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_LPTIM2_OUT

Event 19 triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_LPTIM3_OUT

Event 20 triggers injected group conversion start

ADC_INJECTED_SOFTWARE_START

Software triggers injected group conversion start

ADC Extended Injected Channel Rank**ADC_INJECTED_RANK_1**

ADC injected conversion rank 1

ADC_INJECTED_RANK_2

ADC injected conversion rank 2

ADC_INJECTED_RANK_3

ADC injected conversion rank 3

ADC_INJECTED_RANK_4

ADC injected conversion rank 4

ADC Extended Interrupts Definition**ADC_IT_RDY**

ADC Ready (ADRDY) interrupt source

ADC_IT_EOSMP

ADC End of Sampling interrupt source

ADC_IT_EOC

ADC End of Regular Conversion interrupt source

ADC_IT_EOS

ADC End of Regular sequence of Conversions interrupt source

ADC_IT_OVR

ADC overrun interrupt source

ADC_IT_JEOC

ADC End of Injected Conversion interrupt source

ADC_IT_JEOS

ADC End of Injected sequence of Conversions interrupt source

ADC_IT_AWD1

ADC Analog watchdog 1 interrupt source (main analog watchdog)

ADC_IT_AWD2

ADC Analog watchdog 2 interrupt source (additional analog watchdog)

ADC_IT_AWD3

ADC Analog watchdog 3 interrupt source (additional analog watchdog)

ADC_IT_JQOVF

ADC Injected Context Queue Overflow interrupt source

ADC_IT_AWD

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

ADC Extended Oversampling left Shift**ADC_LEFTBITSHIFT_NONE**

ADC No bit shift

ADC_LEFTBITSHIFT_1

ADC 1 bit shift

ADC_LEFTBITSHIFT_2

ADC 2 bits shift

ADC_LEFTBITSHIFT_3

ADC 3 bits shift

ADC_LEFTBITSHIFT_4

ADC 4 bits shift

ADC_LEFTBITSHIFT_5

ADC 5 bits shift

ADC_LEFTBITSHIFT_6

ADC 6 bits shift

ADC_LEFTBITSHIFT_7

ADC 7 bits shift

ADC_LEFTBITSHIFT_8

ADC 8 bits shift

ADC_LEFTBITSHIFT_9

ADC 9 bits shift

ADC_LEFTBITSHIFT_10

ADC 10 bits shift

ADC_LEFTBITSHIFT_11

ADC 11 bits shift

ADC_LEFTBITSHIFT_12

ADC 12 bits shift

ADC_LEFTBITSHIFT_13

ADC 13 bits shift

ADC_LEFTBITSHIFT_14

ADC 14 bits shift

ADC_LEFTBITSHIFT_15

ADC 15 bits shift

ADC Extended Offset Number**ADC_OFFSET_NONE**

No offset correction

ADC_OFFSET_1

Offset correction to apply to a first channel

ADC_OFFSET_2

Offset correction to apply to a second channel

ADC_OFFSET_3

Offset correction to apply to a third channel

ADC_OFFSET_4

Offset correction to apply to a fourth channel

ADC Extended Regular Oversampling Continued or Resumed Mode**ADC_REGOVERSAMPLING_CONTINUED_MODE**

Oversampling buffer maintained during injection sequence

ADC_REGOVERSAMPLING_RESUMED_MODE

Oversampling buffer zeroed during injection sequence

ADC Extended Oversampling Right Shift**ADC_RIGHTBITSHIFT_NONE**

ADC No bit shift for oversampling

ADC_RIGHTBITSHIFT_1

ADC 1 bit shift for oversampling

ADC_RIGHTBITSHIFT_2

ADC 2 bits shift for oversampling

ADC_RIGHTBITSHIFT_3

ADC 3 bits shift for oversampling

ADC_RIGHTBITSHIFT_4

ADC 4 bits shift for oversampling

ADC_RIGHTBITSHIFT_5

ADC 5 bits shift for oversampling

ADC_RIGHTBITSHIFT_6

ADC 6 bits shift for oversampling

ADC_RIGHTBITSHIFT_7

ADC 7 bits shift for oversampling

ADC_RIGHTBITSHIFT_8

ADC 8 bits shift for oversampling

ADC_RIGHTBITSHIFT_9

ADC 9 bits shift for oversampling

ADC_RIGHTBITSHIFT_10

ADC 10 bits shift for oversampling

ADC_RIGHTBITSHIFT_11

ADC 11 bits shift for oversampling

ADC Extended Single-ended/Differential input mode**ADC_SINGLE_ENDED**

ADC channel set in single-ended input mode

ADC_DIFFERENTIAL_ENDED

ADC channel set in differential mode

ADC Extended Triggered Regular Oversampling**ADC_TRIGGEREDMODE_SINGLE_TRIGGER**

A single trigger for all channel oversampled conversions

ADC_TRIGGEREDMODE_MULTI_TRIGGER

A trigger for each oversampled conversion

7 HAL CEC Generic Driver

7.1 CEC Firmware driver registers structures

7.1.1 CEC_InitTypeDef

Data Fields

- `uint32_t SignalFreeTime`
- `uint32_t Tolerance`
- `uint32_t BRERxStop`
- `uint32_t BREErrorBitGen`
- `uint32_t LBPEErrorBitGen`
- `uint32_t BroadcastMsgNoErrorBitGen`
- `uint32_t SignalFreeTimeOption`
- `uint32_t ListenMode`
- `uint16_t OwnAddress`
- `uint8_t * RxBuffer`

Field Documentation

- `uint32_t CEC_InitTypeDef::SignalFreeTime`

Set SFT field, specifies the Signal Free Time. It can be one of **CEC Signal Free Time setting parameter** and belongs to the set {0,...,7} where 0x0 is the default configuration else means 0.5 + (SignalFreeTime - 1) nominal data bit periods

- `uint32_t CEC_InitTypeDef::Tolerance`

Set RXTOL bit, specifies the tolerance accepted on the received waveforms, it can be a value of **CEC Receiver Tolerance** : it is either CEC_STANDARD_TOLERANCE or CEC_EXTENDED_TOLERANCE

- `uint32_t CEC_InitTypeDef::BRERxStop`

Set BRESTOP bit **CEC Reception Stop on Error** : specifies whether or not a Bit Rising Error stops the reception.

- CEC_NO_RX_STOP_ON_BRE: reception is not stopped.
- CEC_RX_STOP_ON_BRE: reception is stopped.

- `uint32_t CEC_InitTypeDef::BREErrorBitGen`

Set BREGEN bit **CEC Error Bit Generation if Bit Rise Error reported** : specifies whether or not an Error-Bit is generated on the CEC line upon Bit Rising Error detection.

- CEC_BRE_ERRORBIT_NO_GENERATION: no error-bit generation.
- CEC_BRE_ERRORBIT_GENERATION: error-bit generation if BRESTOP is set.

- `uint32_t CEC_InitTypeDef::LBPEErrorBitGen`

Set LBPEGEN bit **CEC Error Bit Generation if Long Bit Period Error reported** : specifies whether or not an Error-Bit is generated on the CEC line upon Long Bit Period Error detection.

- CEC_LBPE_ERRORBIT_NO_GENERATION: no error-bit generation.
- CEC_LBPE_ERRORBIT_GENERATION: error-bit generation.

- `uint32_t CEC_InitTypeDef::BroadcastMsgNoErrorBitGen`

Set BRDNOGEN bit **CEC Error Bit Generation on Broadcast message** : allows to avoid an Error-Bit generation on the CEC line upon an error detected on a broadcast message. It supersedes BREGEN and LBPEGEN bits for a broadcast message error handling. It can take two values:

- CEC_BROADCASTERROR_ERRORBIT_GENERATION.

- a) BRE detection: error-bit generation on the CEC line if BRESTOP=CEC_RX_STOP_ON_BRE and BREGEN=CEC_BRE_ERRORBIT_NO_GENERATION.
- b) LBPE detection: error-bit generation on the CEC line if LBPGEN=CEC_LBPE_ERRORBIT_NO_GENERATION.
- CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION. no error-bit generation in case neither a) nor b) are satisfied. Additionally, there is no error-bit generation in case of Short Bit Period Error detection in a broadcast message while LSTN bit is set.
- **`uint32_t CEC_InitTypeDef::SignalFreeTimeOption`**
Set SFTOP bit ***CEC Signal Free Time start option*** : specifies when SFT timer starts.
 - CEC_SFT_START_ON_TXSOM SFT: timer starts when TXSOM is set by software.
 - CEC_SFT_START_ON_TX_RX_END: SFT timer starts automatically at the end of message transmission/reception.
- **`uint32_t CEC_InitTypeDef::ListenMode`**
Set LSTN bit ***CEC Listening mode option*** : specifies device listening mode. It can take two values:
 - CEC_REDUCED_LISTENING_MODE: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.
 - CEC_FULL_LISTENING_MODE: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.
- **`uint16_t CEC_InitTypeDef::OwnAddress`**
Own addresses configuration This parameter can be a value of ***CEC Own Address***
- **`uint8_t* CEC_InitTypeDef::RxBuffer`**
CEC Rx buffer pointer

7.1.2 CEC_HandleTypeDef

Data Fields

- **`CEC_TypeDef * Instance`**
- **`CEC_InitTypeDef Init`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferCount`**
- **`uint16_t RxXferSize`**
- **`HAL_LockTypeDef Lock`**
- **`HAL_CEC_StateTypeDef gState`**
- **`HAL_CEC_StateTypeDef RxState`**
- **`uint32_t ErrorCode`**

Field Documentation

- **`CEC_TypeDef* CEC_HandleTypeDef::Instance`**
CEC registers base address
- **`CEC_InitTypeDef CEC_HandleTypeDef::Init`**
CEC communication parameters
- **`uint8_t* CEC_HandleTypeDef::pTxBuffPtr`**
Pointer to CEC Tx transfer Buffer
- **`uint16_t CEC_HandleTypeDef::TxXferCount`**
CEC Tx Transfer Counter
- **`uint16_t CEC_HandleTypeDef::RxXferSize`**

- CEC Rx Transfer size, 0: header received only
- **HAL_LockTypeDef CEC_HandleTypeDef::Lock**
Locking object
- **HAL_CEC_StateTypeDef CEC_HandleTypeDef::gState**
CEC state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_CEC_StateTypeDef**
- **HAL_CEC_StateTypeDef CEC_HandleTypeDef::RxState**
CEC state information related to Rx operations. This parameter can be a value of **HAL_CEC_StateTypeDef**
- **uint32_t CEC_HandleTypeDef::ErrorCode**
For errors handling purposes, copy of ISR register in case error is reported

7.2 CEC Firmware driver API description

7.2.1 How to use this driver

The CEC HAL driver can be used as follow:

1. Declare a CEC_HandleTypeDef handle structure.
2. Initialize the CEC low level resources by implementing the HAL_CEC_MspInit ()API:
 - a. Enable the CEC interface clock.
 - b. CEC pins configuration:
 - Enable the clock for the CEC GPIOs.
 - Configure these CEC pins as alternate function pull-up.
 - c. NVIC configuration if you need to use interrupt process (HAL_CEC_Transmit_IT() and HAL_CEC_Receive_IT() APIs):
 - Configure the CEC interrupt priority.
 - Enable the NVIC CEC IRQ handle.
 - The specific CEC interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_CEC_ENABLE_IT() and __HAL_CEC_DISABLE_IT() inside the transmit and receive process.
3. Program the Signal Free Time (SFT) and SFT option, Tolerance, reception stop in case of Bit Rising Error, Error-Bit generation conditions, device logical address and Listen mode in the hcec Init structure.
4. Initialize the CEC registers by calling the HAL_CEC_Init() API.

Note:

This API (HAL_CEC_Init()) configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_CEC_MspInit() API.

7.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the CEC

- The following parameters need to be configured:
 - SignalFreeTime
 - Tolerance
 - BRERxStop (RX stopped or not upon Bit Rising Error)
 - BRErrorBitGen (Error-Bit generation in case of Bit Rising Error)
 - LBPErrorBitGen (Error-Bit generation in case of Long Bit Period Error)
 - BroadcastMsgNoErrorBitGen (Error-bit generation in case of broadcast message error)
 - SignalFreeTimeOption (SFT Timer start definition)
 - OwnAddress (CEC device address)
 - ListenMode

This section contains the following APIs:

- [*HAL_CEC_Init*](#)
- [*HAL_CEC_DelInit*](#)
- [*HAL_CEC_SetDeviceAddress*](#)
- [*HAL_CEC_MspInit*](#)
- [*HAL_CEC_MspDelInit*](#)

7.2.3 IO operation functions

This section contains the following APIs:

- [*HAL_CEC_Transmit_IT*](#)
- [*HAL_CEC_GetLastReceivedFrameSize*](#)
- [*HAL_CEC_ChangeRxBuffer*](#)
- [*HAL_CEC_IRQHandler*](#)
- [*HAL_CEC_TxCpltCallback*](#)
- [*HAL_CEC_RxCpltCallback*](#)
- [*HAL_CEC_ErrorCallback*](#)

7.2.4 Peripheral Control function

This subsection provides a set of functions allowing to control the CEC.

- `HAL_CEC_GetState()` API can be helpful to check in run-time the state of the CEC peripheral.
- `HAL_CEC_GetError()` API can be helpful to check in run-time the error of the CEC peripheral.

This section contains the following APIs:

- [*HAL_CEC_GetState*](#)
- [*HAL_CEC_GetError*](#)

7.2.5 Detailed description of functions

`HAL_CEC_Init`

Function name

`HAL_StatusTypeDef HAL_CEC_Init (CEC_HandleTypeDef * hcec)`

Function description

Initializes the CEC mode according to the specified parameters in the `CEC_InitTypeDef` and creates the associated handle.

Parameters

- **hcec:** CEC handle

Return values

- **HAL:** status

`HAL_CEC_DelInit`

Function name

`HAL_StatusTypeDef HAL_CEC_DelInit (CEC_HandleTypeDef * hcec)`

Function description

Deinitializes the CEC peripheral.

Parameters

- **hcec:** CEC handle

Return values

- **HAL:** status

HAL_CEC_SetDeviceAddress

Function name

HAL_StatusTypeDef HAL_CEC_SetDeviceAddress (CEC_HandleTypeDef * hcec, uint16_t CEC_OwnAddress)

Function description

Initializes the Own Address of the CEC device.

Parameters

- **hcec:** CEC handle
- **CEC_OwnAddress:** The CEC own address.

Return values

- **HAL:** status

HAL_CEC_MspInit

Function name

void HAL_CEC_MspInit (CEC_HandleTypeDef * hcec)

Function description

CEC MSP Init.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_MspDeInit

Function name

void HAL_CEC_MspDeInit (CEC_HandleTypeDef * hcec)

Function description

CEC MSP DeInit.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_Transmit_IT

Function name

HAL_StatusTypeDef HAL_CEC_Transmit_IT (CEC_HandleTypeDef * hcec, uint8_t InitiatorAddress, uint8_t DestinationAddress, uint8_t * pData, uint32_t Size)

Function description

Send data in interrupt mode.

Parameters

- **hcec:** CEC handle
- **InitiatorAddress:** Initiator address
- **DestinationAddress:** destination logical address
- **pData:** pointer to input byte data buffer
- **Size:** amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).

Return values

- **HAL:** status

HAL_CEC_GetLastReceivedFrameSize

Function name

```
uint32_t HAL_CEC_GetLastReceivedFrameSize (CEC_HandleTypeDef * hcec)
```

Function description

Get size of the received frame.

Parameters

- **hcec:** CEC handle

Return values

- **Frame:** size

HAL_CEC_ChangeRxBuffer

Function name

```
void HAL_CEC_ChangeRxBuffer (CEC_HandleTypeDef * hcec, uint8_t * Rxbuffer)
```

Function description

Change Rx Buffer.

Parameters

- **hcec:** CEC handle
- **Rxbuffer:** Rx Buffer

Return values

- **Frame:** size

Notes

- This function can be called only inside the HAL_CEC_RxCpltCallback()

HAL_CEC_IRQHandler

Function name

```
void HAL_CEC_IRQHandler (CEC_HandleTypeDef * hcec)
```

Function description

This function handles CEC interrupt requests.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_TxCpltCallback

Function name

void HAL_CEC_TxCpltCallback (CEC_HandleTypeDef * hcec)

Function description

Tx Transfer completed callback.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_RxCpltCallback

Function name

void HAL_CEC_RxCpltCallback (CEC_HandleTypeDef * hcec, uint32_t RxFrameSize)

Function description

Rx Transfer completed callback.

Parameters

- **hcec:** CEC handle
- **RxFrameSize:** Size of frame

Return values

- **None:**

HAL_CEC_ErrorCallback

Function name

void HAL_CEC_ErrorCallback (CEC_HandleTypeDef * hcec)

Function description

CEC error callbacks.

Parameters

- **hcec:** CEC handle

Return values

- **None:**

HAL_CEC_GetState

Function name

HAL_CEC_StateTypeDef HAL_CEC_GetState (CEC_HandleTypeDef * hcec)

Function description

return the CEC state

Parameters

- **hcec:** pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC module.

Return values

- **HAL:** state

HAL_CEC_GetError**Function name****uint32_t HAL_CEC_GetError (CEC_HandleTypeDef * hcec)****Function description**

Return the CEC error code.

Parameters

- **hcec:** : pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC.

Return values

- **CEC:** Error Code

7.3 CEC Firmware driver defines

7.3.1 CEC

CEC all RX or TX errors flags**CEC_ISR_ALL_ERROR**

CEC Error Bit Generation if Bit Rise Error reported

CEC_BRE_ERRORBIT_NO_GENERATION**CEC_BRE_ERRORBIT_GENERATION**

CEC Reception Stop on Error

CEC_NO_RX_STOP_ON_BRE**CEC_RX_STOP_ON_BRE**

CEC Error Bit Generation on Broadcast message

CEC_BROADCASTERROR_ERRORBIT_GENERATION**CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION**

CEC Error Code

HAL_CEC_ERROR_NONE

no error

HAL_CEC_ERROR_RXOVR

CEC Rx-Overrun

HAL_CEC_ERROR_BRE

CEC Rx Bit Rising Error

HAL_CEC_ERROR_SBPE

CEC Rx Short Bit period Error

HAL_CEC_ERROR_LBPE

CEC Rx Long Bit period Error

HAL_CEC_ERROR_RXACKE

CEC Rx Missing Acknowledge

HAL_CEC_ERROR_ARBLST

CEC Arbitration Lost

HAL_CEC_ERROR_TXUDR

CEC Tx-Buffer Underrun

HAL_CEC_ERROR_TXERR

CEC Tx-Error

HAL_CEC_ERROR_TXACKE

CEC Tx Missing Acknowledge

CEC Exported Macros**_HAL_CEC_RESET_HANDLE_STATE****Description:**

- Reset CEC handle gstate & RxState.

Parameters:

- _HANDLE_: CEC handle.

Return value:

- None

_HAL_CEC_GET_FLAG**Description:**

- Checks whether or not the specified CEC interrupt flag is set.

Parameters:

- _HANDLE_: specifies the CEC Handle.
- _FLAG_: specifies the flag to check.
 - CEC_FLAG_TXACKE: Tx Missing acknowledge Error
 - CEC_FLAG_TXERR: Tx Error.
 - CEC_FLAG_TXUDR: Tx-Buffer Underrun.
 - CEC_FLAG_TXEND: End of transmission (successful transmission of the last byte).
 - CEC_FLAG_TXBR: Tx-Byte Request.
 - CEC_FLAG_ARBLST: Arbitration Lost
 - CEC_FLAG_RXACKE: Rx-Missing Acknowledge
 - CEC_FLAG_LBPE: Rx Long period Error
 - CEC_FLAG_SBPE: Rx Short period Error
 - CEC_FLAG_BRE: Rx Bit Rising Error
 - CEC_FLAG_RXOVR: Rx Overrun.
 - CEC_FLAG_RXEND: End Of Reception.
 - CEC_FLAG_RXBR: Rx-Byte Received.

Return value:

- ITStatus

__HAL_CEC_CLEAR_FLAG

Description:

- Clears the interrupt or status flag when raised (write at 1)

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the interrupt/status flag to clear. This parameter can be one of the following values:
 - CEC_FLAG_TXACKE: Tx Missing acknowledge Error
 - CEC_FLAG_TXERR: Tx Error.
 - CEC_FLAG_RXUDR: Rx-Buffer Underrun.
 - CEC_FLAG_TXEND: End of transmission (successful transmission of the last byte).
 - CEC_FLAG_RXBR: Rx-Byte Request.
 - CEC_FLAG_ARBLST: Arbitration Lost
 - CEC_FLAG_RXACKE: Rx-Missing Acknowledge
 - CEC_FLAG_LBPE: Rx Long period Error
 - CEC_FLAG_SBPE: Rx Short period Error
 - CEC_FLAG_BRE: Rx Bit Rising Error
 - CEC_FLAG_RXOVR: Rx Overrun.
 - CEC_FLAG_RXEND: End Of Reception.
 - CEC_FLAG_RXBR: Rx-Byte Received.

Return value:

- none

__HAL_CEC_ENABLE_IT

Description:

- Enables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to enable. This parameter can be one of the following values:
 - CEC_IT_TXACKE: Tx Missing acknowledge Error IT Enable
 - CEC_IT_TXERR: Tx Error IT Enable
 - CEC_IT_RXUDR: Rx-Buffer Underrun IT Enable
 - CEC_IT_RXEND: End of transmission IT Enable
 - CEC_IT_RXBR: Rx-Byte Request IT Enable
 - CEC_IT_ARBLST: Arbitration Lost IT Enable
 - CEC_IT_RXACKE: Rx-Missing Acknowledge IT Enable
 - CEC_IT_LBPE: Rx Long period Error IT Enable
 - CEC_IT_SBPE: Rx Short period Error IT Enable
 - CEC_IT_BRE: Rx Bit Rising Error IT Enable
 - CEC_IT_RXOVR: Rx Overrun IT Enable
 - CEC_IT_RXEND: End Of Reception IT Enable
 - CEC_IT_RXBR: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_DISABLE_IT

Description:

- Disables the specified CEC interrupt.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.
- `__INTERRUPT__`: specifies the CEC interrupt to disable. This parameter can be one of the following values:
 - `CEC_IT_TXACKE`: Tx Missing acknowledge Error IT Enable
 - `CEC_IT_RXERR`: Tx Error IT Enable
 - `CEC_IT_TXUDR`: Tx-Buffer Underrun IT Enable
 - `CEC_IT_TXEND`: End of transmission IT Enable
 - `CEC_IT_TXBR`: Tx-Byte Request IT Enable
 - `CEC_IT_ARBLST`: Arbitration Lost IT Enable
 - `CEC_IT_RXACKE`: Rx-Missing Acknowledge IT Enable
 - `CEC_IT_LBPE`: Rx Long period Error IT Enable
 - `CEC_IT_SBPE`: Rx Short period Error IT Enable
 - `CEC_IT_BRE`: Rx Bit Rising Error IT Enable
 - `CEC_IT_RXOVR`: Rx Overrun IT Enable
 - `CEC_IT_RXEND`: End Of Reception IT Enable
 - `CEC_IT_RXBR`: Rx-Byte Received IT Enable

Return value:

- none

__HAL_CEC_GET_IT_SOURCE

Description:

- Checks whether or not the specified CEC interrupt is enabled.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.
- `__INTERRUPT__`: specifies the CEC interrupt to check. This parameter can be one of the following values:
 - `CEC_IT_TXACKE`: Tx Missing acknowledge Error IT Enable
 - `CEC_IT_RXERR`: Tx Error IT Enable
 - `CEC_IT_TXUDR`: Tx-Buffer Underrun IT Enable
 - `CEC_IT_TXEND`: End of transmission IT Enable
 - `CEC_IT_TXBR`: Tx-Byte Request IT Enable
 - `CEC_IT_ARBLST`: Arbitration Lost IT Enable
 - `CEC_IT_RXACKE`: Rx-Missing Acknowledge IT Enable
 - `CEC_IT_LBPE`: Rx Long period Error IT Enable
 - `CEC_IT_SBPE`: Rx Short period Error IT Enable
 - `CEC_IT_BRE`: Rx Bit Rising Error IT Enable
 - `CEC_IT_RXOVR`: Rx Overrun IT Enable
 - `CEC_IT_RXEND`: End Of Reception IT Enable
 - `CEC_IT_RXBR`: Rx-Byte Received IT Enable

Return value:

- `FlagStatus`

__HAL_CEC_ENABLE

Description:

- Enables the CEC device.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.

Return value:

- none

[__HAL_CEC_DISABLE](#)**Description:**

- Disables the CEC device.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

[__HAL_CEC_FIRST_BYTE_TX_SET](#)**Description:**

- Set Transmission Start flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none

[__HAL_CEC_LAST_BYTE_TX_SET](#)**Description:**

- Set Transmission End flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none: If the CEC message consists of only one byte, TXEOM must be set before of TXSOM.

[__HAL_CEC_GET_TRANSMISSION_START_FLAG](#)**Description:**

- Get Transmission Start flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- FlagStatus

[__HAL_CEC_GET_TRANSMISSION_END_FLAG](#)**Description:**

- Get Transmission End flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- FlagStatus

[__HAL_CEC_CLEAR_OAR](#)**Description:**

- Clear OAR register.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.

Return value:

- none

[_HAL_CEC_SET_OAR](#)**Description:**

- Set OAR register (without resetting previously set address in case of multi-address mode) To reset OAR,

Parameters:

- `__HANDLE__`: specifies the CEC Handle.
- `__ADDRESS__`: Own Address value (CEC logical address is identified by bit position)

Return value:

- none

CEC Flags definition**[CEC_FLAG_TXACKE](#)****[CEC_FLAG_TXERR](#)****[CEC_FLAG_TXUDR](#)****[CEC_FLAG_TXEND](#)****[CEC_FLAG_TXBR](#)****[CEC_FLAG_ARBLST](#)****[CEC_FLAG_RXACKE](#)****[CEC_FLAG_LBPE](#)****[CEC_FLAG_SBPE](#)****[CEC_FLAG_BRE](#)****[CEC_FLAG_RXOVR](#)****[CEC_FLAG_RXEND](#)****[CEC_FLAG_RXBR](#)*****CEC all RX errors interrupts enabling flag*****[CEC_IER_RX_ALL_ERR](#)*****CEC all TX errors interrupts enabling flag*****[CEC_IER_TX_ALL_ERR](#)*****CEC Initiator logical address position in message header*****[CEC_INITIATOR_LSB_POS](#)*****CEC Interrupts definition***

CEC_IT_TXACKE

CEC_IT_TXERR

CEC_IT_RXUDR

CEC_IT_TXEND

CEC_IT_TXBR

CEC_IT_ARBLST

CEC_IT_RXACKE

CEC_IT_LBPE

CEC_IT_SBPE

CEC_IT_BRE

CEC_IT_RXOVR

CEC_IT_RXEND

CEC_IT_RXBR

CEC Error Bit Generation if Long Bit Period Error reported

CEC_LBPE_ERRORBIT_NO_GENERATION

CEC_LBPE_ERRORBIT_GENERATION

CEC Listening mode option

CEC_REDUCED_LISTENING_MODE

CEC_FULL_LISTENING_MODE

CEC Device Own Address position in CEC CFGR register

CEC_CFGR_OAR_LSB_POS

CEC Own Address

CEC_OWN_ADDRESS_NONE

CEC_OWN_ADDRESS_0

CEC_OWN_ADDRESS_1

CEC_OWN_ADDRESS_2

CEC_OWN_ADDRESS_3

CEC_OWN_ADDRESS_4

CEC_OWN_ADDRESS_5

CEC_OWN_ADDRESS_6

CEC_OWN_ADDRESS_7

CEC_OWN_ADDRESS_8

CEC_OWN_ADDRESS_9

CEC_OWN_ADDRESS_10

CEC_OWN_ADDRESS_11

CEC_OWN_ADDRESS_12

CEC_OWN_ADDRESS_13

CEC_OWN_ADDRESS_14

CEC Signal Free Time start option

CEC_SFT_START_ON_TXSOM

CEC_SFT_START_ON_TX_RX_END

CEC Signal Free Time setting parameter

CEC_DEFAULT_SFT

CEC_0_5_BITPERIOD_SFT

CEC_1_5_BITPERIOD_SFT

CEC_2_5_BITPERIOD_SFT

CEC_3_5_BITPERIOD_SFT

CEC_4_5_BITPERIOD_SFT

CEC_5_5_BITPERIOD_SFT

CEC_6_5_BITPERIOD_SFT

CEC Receiver Tolerance

CEC_STANDARD_TOLERANCE

CEC_EXTENDED_TOLERANCE

8 HAL COMP Generic Driver

8.1 COMP Firmware driver registers structures

8.1.1 COMP_InitTypeDef

Data Fields

- *uint32_t WindowMode*
- *uint32_t Mode*
- *uint32_t NonInvertingInput*
- *uint32_t InvertingInput*
- *uint32_t Hysteresis*
- *uint32_t OutputPol*
- *uint32_t BlankingSrce*
- *uint32_t TriggerMode*

Field Documentation

- *uint32_t COMP_InitTypeDef::WindowMode*

Set window mode of a pair of comparators instances (2 consecutive instances odd and even COMP<x> and COMP<x+1>). Note: HAL COMP driver allows to set window mode from any COMP instance of the pair of COMP instances composing window mode. This parameter can be a value of **COMP Window Mode**

- *uint32_t COMP_InitTypeDef::Mode*

Set comparator operating mode to adjust power and speed. Note: For the characteristics of comparator power modes (propagation delay and power consumption), refer to device datasheet. This parameter can be a value of **COMP power mode**

- *uint32_t COMP_InitTypeDef::NonInvertingInput*

Set comparator input plus (non-inverting input). This parameter can be a value of **COMP input plus (non-inverting input)**

- *uint32_t COMP_InitTypeDef::InvertingInput*

Set comparator input minus (inverting input). This parameter can be a value of **COMP input minus (inverting input)**

- *uint32_t COMP_InitTypeDef::Hysteresis*

Set comparator hysteresis mode of the input minus. This parameter can be a value of **COMP hysteresis**

- *uint32_t COMP_InitTypeDef::OutputPol*

Set comparator output polarity. This parameter can be a value of **COMP Output Polarity**

- *uint32_t COMP_InitTypeDef::BlankingSrce*

Set comparator blanking source. This parameter can be a value of **COMP Blanking Source**

- *uint32_t COMP_InitTypeDef::TriggerMode*

Set the comparator output triggering External Interrupt Line (EXTI). This parameter can be a value of **COMP output to EXTI**

8.1.2 COMP_HandleTypeDef

Data Fields

- *COMP_TypeDef * Instance*

- ***COMP_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***_IO HAL_COMP_StateTypeDef State***

Field Documentation

- ***COMP_TypeDef* COMP_HandleTypeDef::Instance***
Register base address
- ***COMP_InitTypeDef COMP_HandleTypeDef::Init***
COMP required parameters
- ***HAL_LockTypeDef COMP_HandleTypeDef::Lock***
Locking object
- ***_IO HAL_COMP_StateTypeDef COMP_HandleTypeDef::State***
COMP communication state

8.2 COMP Firmware driver API description

8.2.1 COMP Peripheral features

The STM32H7xx device family integrates two analog comparators instances COMP1 and COMP2:

1. The COMP input minus (inverting input) and input plus (non inverting input) can be set to internal references or to GPIO pins (refer to GPIO list in reference manual).
2. The COMP output level is available using `HAL_COMP_GetOutputLevel()` and can be redirected to other peripherals: GPIO pins (in mode alternate functions for comparator), timers. (refer to GPIO list in reference manual).
3. Pairs of comparators instances can be combined in window mode (2 consecutive instances odd and even `COMP<x>` and `COMP<x+1>`).
4. The comparators have interrupt capability through the EXTI controller with wake-up from sleep and stop modes:
 - COMP1 is internally connected to EXTI Line 20
 - COMP2 is internally connected to EXTI Line 21

From the corresponding IRQ handler, the right interrupt source can be retrieved using macro `__HAL_COMP_COMP1_EXTI_GET_FLAG()` and `__HAL_COMP_COMP2_EXTI_GET_FLAG()`.

8.2.2 How to use this driver

This driver provides functions to configure and program the comparator instances of STM32H7xx devices. To use the comparator, perform the following steps:

1. Initialize the COMP low level resources by implementing the `HAL_COMP_MspInit()`:
 - Configure the GPIO connected to comparator inputs plus and minus in analog mode using `HAL_GPIO_Init()`.
 - If needed, configure the GPIO connected to comparator output in alternate function mode using `HAL_GPIO_Init()`.
 - If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using `HAL_GPIO_Init()` function. After that enable the comparator interrupt vector using `HAL_NVIC_EnableIRQ()` function.
2. Configure the comparator using `HAL_COMP_Init()` function:
 - Select the input minus (inverting input)
 - Select the input plus (non-inverting input)
 - Select the hysteresis
 - Select the blanking source

- Select the output polarity
- Select the power mode
- Select the window mode

Note:

HAL_COMP_Init() calls internally __HAL_RCC_SYSCFG_CLK_ENABLE() to enable internal control clock of the comparators. However, this is a legacy strategy. Therefore, for compatibility anticipation, it is recommended to implement __HAL_RCC_SYSCFG_CLK_ENABLE() in "HAL_COMP_MspInit()". In STM32H7, COMP clock enable __HAL_RCC_COMP12_CLK_ENABLE() must be implemented by user in "HAL_COMP_MspInit()".

3. Reconfiguration on-the-fly of comparator can be done by calling again function HAL_COMP_Init() with new input structure parameters values.
4. Enable the comparator using HAL_COMP_Start() or HAL_COMP_Start_IT() to be enabled with the interrupt through NVIC of the CPU. Note: HAL_COMP_Start_IT() must be called after each interrupt otherwise the interrupt mode will stay disabled.
5. Use HAL_COMP_GetOutputLevel() or HAL_COMP_TriggerCallback() functions to manage comparator outputs(output level or events)
6. Disable the comparator using HAL_COMP_Stop() or HAL_COMP_Stop_IT() to disable the interrupt too.
7. De-initialize the comparator using HAL_COMP_DelInit() function.
8. For safety purpose, comparator configuration can be locked using HAL_COMP_Lock() function. The only way to unlock the comparator is a device hardware reset.

8.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- [**HAL_COMP_Init**](#)
- [**HAL_COMP_DelInit**](#)
- [**HAL_COMP_MspInit**](#)
- [**HAL_COMP_MspDelInit**](#)

8.2.4 IO operation functions

This section provides functions allowing to:

- Start a Comparator instance without interrupt.
- Stop a Comparator instance without interrupt.
- Start a Comparator instance with interrupt generation.
- Stop a Comparator instance with interrupt generation.

This section contains the following APIs:

- [**HAL_COMP_Start**](#)
- [**HAL_COMP_Stop**](#)
- [**HAL_COMP_Start_IT**](#)
- [**HAL_COMP_Stop_IT**](#)
- [**HAL_COMP_IRQHandler**](#)

8.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the comparators.

This section contains the following APIs:

- [**HAL_COMP_Lock**](#)
- [**HAL_COMP_GetOutputLevel**](#)
- [**HAL_COMP_TriggerCallback**](#)

8.2.6 Peripheral State functions

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

This section contains the following APIs:

- [HAL_COMP_GetState](#)

8.2.7 Detailed description of functions

HAL_COMP_Init

Function name

`HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)`

Function description

Initialize the COMP according to the specified parameters in the COMP_InitTypeDef and initialize the associated handle.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_COMP_DeInit

Function name

`HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)`

Function description

Deinitialize the COMP peripheral.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- Deinitialization cannot be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

HAL_COMP_MspInit

Function name

`void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)`

Function description

Initialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_MspDelInit

Function name

void HAL_COMP_MspDelInit (COMP_HandleTypeDef * hcomp)

Function description

Deinitialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_Start

Function name

HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)

Function description

Start the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Stop

Function name

HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)

Function description

Stop the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Start_IT

Function name

HAL_StatusTypeDef HAL_COMP_Start_IT (COMP_HandleTypeDef * hcomp)

Function description

Enable the interrupt and start the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Stop_IT**Function name**

```
HAL_StatusTypeDef HAL_COMP_Stop_IT (COMP_HandleTypeDef * hcomp)
```

Function description

Disable the interrupt and Stop the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_IRQHandler**Function name**

```
void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)
```

Function description

Comparator IRQ Handler.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Lock**Function name**

```
HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)
```

Function description

Lock the selected comparator configuration.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- A system reset is required to unlock the comparator configuration.

HAL_COMP_GetOutputLevel**Function name**

```
uint32_t HAL_COMP_GetOutputLevel (COMP_HandleTypeDef * hcomp)
```

Function description

Return the output level (high or low) of the selected comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **Returns:** the selected comparator output level:
 - COMP_OUTPUT_LEVEL_LOW
 - COMP_OUTPUT_LEVEL_HIGH

Notes

- The output level depends on the selected polarity. If the polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minus. Comparator output is high when the input plus is at a higher voltage than the input minus. If the polarity is inverted: Comparator output is high when the input plus is at a lower voltage than the input minus. Comparator output is low when the input plus is at a higher voltage than the input minus.

HAL_COMP_TriggerCallback

Function name

void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)

Function description

Comparator callback.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_GetState

Function name

HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)

Function description

Return the COMP handle state.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** state

8.3 COMP Firmware driver defines

8.3.1 COMP

COMP Blanking Source

COMP_BLANKINGSRC_NONE

No blanking source

COMP_BLANKINGSRC_TIM1_OC5

TIM1 OC5 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM2_OC3

TIM2 OC3 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM3_OC3

TIM3 OC3 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM3_OC4

TIM3 OC4 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM8_OC5

TIM8 OC5 selected as blanking source for comparator

COMP_BLANKINGSRC_TIM15_OC1

TIM15 OC1 selected as blanking source for comparator

COMP Exported Macros**_HAL_COMP_ENABLE_OR****Description:**

- Enable the specified bit in the Option register.

Parameters:

- AF: specifies the Alternate Function source selection . This parameter can be one of the following values:
 - COMP_OR_AFOPA6 : Alternate Function PA6 source selection
 - COMP_OR_AFOPA8 : Alternate Function PA8 source selection
 - COMP_OR_AFOPB12 : Alternate Function PB12 source selection
 - COMP_OR_AFOPE6 : Alternate Function PE6 source selection
 - COMP_OR_AFOPE15 : Alternate Function PE15 source selection
 - COMP_OR_AFOPG2 : Alternate Function PG2 source selection
 - COMP_OR_AFOPG3 : Alternate Function PG3 source selection
 - COMP_OR_AFOPG4 : Alternate Function PG4 source selection
 - COMP_OR_AFOPI1 : Alternate Function PI1 source selection
 - COMP_OR_AFOPI4 : Alternate Function PI4 source selection
 - COMP_OR_AFOPK2 : Alternate Function PK2 source selection

Return value:

- None

_HAL_COMP_DISABLE_OR**Description:**

- Disable the specified bit in the Option register.

Parameters:

- AF: specifies the Alternate Function source selection . This parameter can be one of the following values:
 - COMP_OR_AFOPA6 : Alternate Function PA6 source selection
 - COMP_OR_AFOPA8 : Alternate Function PA8 source selection
 - COMP_OR_AFOPB12 : Alternate Function PB12 source selection
 - COMP_OR_AFOPE6 : Alternate Function PE6 source selection
 - COMP_OR_AFOPE15 : Alternate Function PE15 source selection
 - COMP_OR_AFOPG2 : Alternate Function PG2 source selection
 - COMP_OR_AFOPG3 : Alternate Function PG3 source selection

- COMP_OR_AFOPG4 : Alternate Function PG4 source selection
- COMP_OR_AFOPI1 : Alternate Function PI1 source selection
- COMP_OR_AFOPI4 : Alternate Function PI4 source selection
- COMP_OR_AFOPK2 : Alternate Function PK2 source selection

Return value:

- None

COMP Exported Types**COMP_STATE_BITFIELD_LOCK****COMP EXTI Lines****COMP_EXTI_LINE_COMP1**

EXTI line 20 connected to COMP1 output

COMP_EXTI_LINE_COMP2

EXTI line 21 connected to COMP2 output

COMP_EXTI_IT

EXTI line event with interruption

COMP_EXTI_EVENT

EXTI line event only (without interruption)

COMP_EXTI_RISING

EXTI line event on rising edge

COMP_EXTI_FALLING

EXTI line event on falling edge

COMP external interrupt line management**_HAL_COMP_COMP1_EXTI_ENABLE_RISING_EDGE****Description:**

- Enable the COMP1 EXTI line rising edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EXTI_DISABLE_RISING_EDGE**Description:**

- Disable the COMP1 EXTI line rising edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EXTI_ENABLE_FALLING_EDGE**Description:**

- Enable the COMP1 EXTI line falling edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EXTI_DISABLE_FALLING_EDGE**Description:**

- Disable the COMP1 EXTI line falling edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EXTI_ENABLE_RISING_FALLING_EDGE**Description:**

- Enable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EXTI_DISABLE_RISING_FALLING_EDGE**Description:**

- Disable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

_HAL_COMP_COMP1_EXTI_ENABLE_IT**Description:**

- Enable the COMP1 EXTI line in interrupt mode.

Return value:

- None

_HAL_COMP_COMP1_EXTI_DISABLE_IT**Description:**

- Disable the COMP1 EXTI line in interrupt mode.

Return value:

- None

_HAL_COMP_COMP1_EXTI_ENABLE_EVENT**Description:**

- Enable the COMP1 EXTI Line in event mode.

Return value:

- None

_HAL_COMP_COMP1_EXTI_DISABLE_EVENT**Description:**

- Disable the COMP1 EXTI Line in event mode.

Return value:

- None

_HAL_COMP_COMP1_EXTI_GET_FLAG**Description:**

- Check whether the COMP1 EXTI line flag is set or not.

Return value:

- RESET: or SET

[__HAL_COMP_COMP1_EXTI_CLEAR_FLAG](#)**Description:**

- Clear the COMP1 EXTI flag.

Return value:

- None

[__HAL_COMP_COMP1_EXTI_GENERATE_SWIT](#)**Description:**

- Generate a software interrupt on the COMP1 EXTI line.

Return value:

- None

[__HAL_COMP_COMP1_EXTID3_ENABLE_EVENT](#)**Description:**

- Enable the COMP1 D3 EXTI Line in event mode.

Return value:

- None

[__HAL_COMP_COMP1_EXTID3_DISABLE_EVENT](#)**Description:**

- Disable the COMP1 D3 EXTI Line in event mode.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_ENABLE_RISING_EDGE](#)**Description:**

- Enable the COMP2 EXTI line rising edge trigger.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_DISABLE_RISING_EDGE](#)**Description:**

- Disable the COMP2 EXTI line rising edge trigger.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_ENABLE_FALLING_EDGE](#)**Description:**

- Enable the COMP2 EXTI line falling edge trigger.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_DISABLE_FALLING_EDGE](#)**Description:**

- Disable the COMP2 EXTI line falling edge trigger.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_ENABLE_RISING_FALLING_EDGE](#)

Description:

- Enable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_DISABLE_RISING_FALLING_EDGE](#)

Description:

- Disable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_ENABLE_IT](#)

Description:

- Enable the COMP2 EXTI line.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_DISABLE_IT](#)

Description:

- Disable the COMP2 EXTI line.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_ENABLE_EVENT](#)

Description:

- Enable the COMP2 EXTI Line in event mode.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_DISABLE_EVENT](#)

Description:

- Disable the COMP2 EXTI Line in event mode.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_GET_FLAG](#)

Description:

- Check whether the COMP2 EXTI line flag is set or not.

Return value:

- RESET: or SET

[__HAL_COMP_COMP2_EXTI_CLEAR_FLAG](#)

Description:

- Clear the the COMP2 EXTI flag.

Return value:

- None

[__HAL_COMP_COMP2_EXTID3_ENABLE_EVENT](#)**Description:**

- Enable the COMP2 D3 EXTI Line in event mode.

Return value:

- None

[__HAL_COMP_COMP2_EXTID3_DISABLE_EVENT](#)**Description:**

- Disable the COMP2 D3 EXTI Line in event mode.

Return value:

- None

[__HAL_COMP_COMP2_EXTI_GENERATE_SWIT](#)**Description:**

- Generate a software interrupt on the COMP2 EXTI line.

Return value:

- None

[__HAL_COMP_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- __HANDLE__: specifies the COMP Handle. This parameter can be COMP1 where x: 1 or 2 to select the COMP peripheral.
- __INTERRUPT__: specifies the COMP interrupt source to check. This parameter can be one of the following values:
 - COMP_IT_EN: Comparator interrupt enable

Return value:

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

[__HAL_COMP_GET_FLAG](#)**Description:**

- Checks whether the specified COMP flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - COMP_FLAG_C1I: Comparator 1 Interrupt Flag
 - COMP_FLAG_C2I: Comparator 2 Interrupt Flag

Return value:

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

[__HAL_COMP_CLEAR_FLAG](#)**Description:**

- Clears the specified COMP pending flag.

Parameters:

- `_FLAG_`: specifies the flag to check. This parameter can be any combination of the following values:
 - `COMP_CLEAR_C1IF` : Clear Comparator 1 Interrupt Flag
 - `COMP_CLEAR_C2IF` : Clear Comparator 2 Interrupt Flag

Return value:

- None

`_HAL_COMP_CLEAR_C1IFLAG`**Description:**

- Clear the COMP C1I flag.

Return value:

- None

`_HAL_COMP_CLEAR_C2IFLAG`**Description:**

- Clear the COMP C2I flag.

Return value:

- None

`_HAL_COMP_ENABLE_IT`**Description:**

- Enable the specified COMP interrupt.

Parameters:

- `_HANDLE_`: specifies the COMP Handle.
- `_INTERRUPT_`: specifies the COMP interrupt source to enable. This parameter can be one of the following values:
 - `COMP_CFGRx_ITEN` : Comparator interrupt

Return value:

- None

`_HAL_COMP_DISABLE_IT`**Description:**

- Disable the specified COMP interrupt.

Parameters:

- `_HANDLE_`: specifies the COMP Handle.
- `_INTERRUPT_`: specifies the COMP interrupt source to enable. This parameter can be one of the following values:
 - `COMP_CFGRx_ITEN` : Comparator interrupt

Return value:

- None

COMP output to EXTI**`COMP_TRIGGERMODE_NONE`**

Comparator output triggering no External Interrupt Line

`COMP_TRIGGERMODE_IT_RISING`

Comparator output triggering External Interrupt Line event with interruption, on rising edge

COMP_TRIGGERMODE_IT_FALLING

Comparator output triggering External Interrupt Line event with interruption, on falling edge

COMP_TRIGGERMODE_IT_RISING_FALLING

Comparator output triggering External Interrupt Line event with interruption, on both rising and falling edges

COMP_TRIGGERMODE_EVENT_RISING

Comparator output triggering External Interrupt Line event only (without interruption), on rising edge

COMP_TRIGGERMODE_EVENT_FALLING

Comparator output triggering External Interrupt Line event only (without interruption), on falling edge

COMP_TRIGGERMODE_EVENT_RISING_FALLING

Comparator output triggering External Interrupt Line event only (without interruption), on both rising and falling edges

COMP Flag**COMP_FLAG_C1I**

Comparator 1 Interrupt Flag

COMP_FLAG_C2I

Comparator 2 Interrupt Flag

COMP_FLAG_LOCK

Lock flag

COMP Private macros to get EXTI line associated with Comparators**COMP_GET_EXTI_LINE****Description:**

- Get the specified EXTI line for a comparator instance.

Parameters:

- __INSTANCE__: specifies the COMP instance.

Return value:

- value: of

COMP Handle Management**_HAL_COMP_RESET_HANDLE_STATE****Description:**

- Reset COMP handle state.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

_HAL_COMP_ENABLE**Description:**

- Enable the specified comparator.

Parameters:

- __HANDLE__: COMP handle

Return value:

- None

_HAL_COMP_DISABLE**Description:**

- Disable the specified comparator.

Parameters:

- `_HANDLE_`: COMP handle

Return value:

- None

_HAL_COMP_LOCK**Description:**

- Lock the specified comparator configuration.

Parameters:

- `_HANDLE_`: COMP handle

Return value:

- None

Notes:

- Using this macro induce HAL COMP handle state machine being no more in line with COMP instance state. To keep HAL COMP handle state machine updated, it is recommended to use function "HAL_COMP_Lock".

_HAL_COMP_IS_LOCKED**Description:**

- Check whether the specified comparator is locked.

Parameters:

- `_HANDLE_`: COMP handle

Return value:

- Value: 0 if COMP instance is not locked, value 1 if COMP instance is locked

COMP hysteresis**COMP_HYSTERESIS_NONE**

No hysteresis

COMP_HYSTERESIS_LOW

Hysteresis level low

COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

COMP_HYSTERESIS_HIGH

Hysteresis level high

COMP input minus (inverting input)**COMP_INPUT_MINUS_1_4VREFINT**

Comparator input minus connected to 1/4 VrefInt

COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)

COMP_INPUT_MINUS_DAC1_CH2

Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)

COMP_INPUT_MINUS_IO1

Comparator input minus connected to IO1 (pin PB1 for COMP1, pin PE10 for COMP2)

COMP_INPUT_MINUS_IO2

Comparator input minus connected to IO2 (pin PC4 for COMP1, pin PE7 for COMP2)

COMP input plus (non-inverting input)**COMP_INPUT_PLUS_IO1**

Comparator input plus connected to IO1 (pin PB0 for COMP1, pin PE9 for COMP2)

COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PB2 for COMP1, pin PE11 for COMP2)

COMP Interrupts Definitions**COMP_IT_EN*****COMP private macros to check input parameters*****IS_COMP_WINDOWMODE****IS_COMP_POWERMODE****IS_COMP_INPUT_PLUS****IS_COMP_INPUT_MINUS****IS_COMP_HYSTERESIS****IS_COMP_OUTPUTPOL****IS_COMP_BLANKINGSRCE****IS_COMP_TRIGGERMODE****IS_COMP_OUTPUT_LEVEL*****COMP Interruption Clear Flags*****COMP_CLEAR_C1IF**

Clear Comparator 1 Interrupt Flag

COMP_CLEAR_C2IF

Clear Comparator 2 Interrupt Flag

COMP Output Level**COMP_OUTPUT_LEVEL_LOW****COMP_OUTPUT_LEVEL_HIGH*****COMP Output Polarity*****COMP_OUTPUTPOL_NONINVERTED**

COMP output level is not inverted (comparator output is high when the input plus is at a higher voltage than the input minus)

COMP_OUTPUTPOL_INVERTED

COMP output level is inverted (comparator output is low when the input plus is at a higher voltage than the input minus)

COMP power mode**COMP_POWERMODE_HIGHSPEED**

High Speed

COMP_POWERMODE_MEDIUMSPEED

Medium Speed

COMP_POWERMODE_ULTRALOWPOWER

Ultra-low power mode

COMP Window Mode**COMP_WINDOWMODE_DISABLE**

Window mode disable: Comparators instances pair COMP1 and COMP2 are independent

COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON

Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

9 HAL CORTEX Generic Driver

9.1 CORTEX Firmware driver registers structures

9.1.1 MPU_Region_InitTypeDef

Data Fields

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

Field Documentation

- `uint8_t MPU_Region_InitTypeDef::Enable`
Specifies the status of the region. This parameter can be a value of **CORTEX MPU Region Enable**
- `uint8_t MPU_Region_InitTypeDef::Number`
Specifies the number of the region to protect. This parameter can be a value of **CORTEX MPU Region Number**
- `uint32_t MPU_Region_InitTypeDef::BaseAddress`
Specifies the base address of the region to protect.
- `uint8_t MPU_Region_InitTypeDef::Size`
Specifies the size of the region to protect. This parameter can be a value of **CORTEX MPU Region Size**
- `uint8_t MPU_Region_InitTypeDef::SubRegionDisable`
Specifies the number of the subregion protection to disable. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- `uint8_t MPU_Region_InitTypeDef::TypeExtField`
Specifies the TEX field level. This parameter can be a value of **MPU TEX Levels**
- `uint8_t MPU_Region_InitTypeDef::AccessPermission`
Specifies the region access permission type. This parameter can be a value of **CORTEX MPU Region Permission Attributes**
- `uint8_t MPU_Region_InitTypeDef::DisableExec`
Specifies the instruction access status. This parameter can be a value of **CORTEX MPU Instruction Access**
- `uint8_t MPU_Region_InitTypeDef::IsShareable`
Specifies the shareability status of the protected region. This parameter can be a value of **CORTEX MPU Instruction Access Shareable**
- `uint8_t MPU_Region_InitTypeDef::IsCacheable`

Specifies the cacheable status of the region protected. This parameter can be a value of **CORTEX MPU Instruction Access Cacheable**

- `uint8_t MPU_Region_InitTypeDef::IsBufferable`

Specifies the bufferable status of the protected region. This parameter can be a value of **CORTEX MPU Instruction Access Bufferable**

9.2 CORTEX Firmware driver API description

9.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

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

1. Configure the NVIC Priority Grouping using `HAL_NVIC_SetPriorityGrouping()` function according to the following table.
2. Configure the priority of the selected IRQ Channels using `HAL_NVIC_SetPriority()`.
3. Enable the selected IRQ Channels using `HAL_NVIC_EnableIRQ()`.
4. please refer to programming manual for details in how to configure priority.

Note: When the `NVIC_PRIORITYGROUP_0` is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the sub priority.

Note: IRQ priority order (sorted by highest to lowest priority):

- Lowest preemption priority
- Lowest sub priority
- Lowest hardware priority (IRQ number)

How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The `HAL_SYSTICK_Config()` function calls the `SysTick_Config()` function which is a CMSIS function that:
 - Configures the SysTick Reload register with value passed as function parameter.
 - Configures the SysTick IRQ priority to the lowest value (0x0F).
 - Resets the SysTick Counter register.
 - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
 - Enables the SysTick Interrupt.
 - Starts the SysTick Counter.
- You can change the SysTick Clock source to be `HCLK_Div8` by calling the macro `HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK_DIV8)` just after the `HAL_SYSTICK_Config()` function call. The `HAL_SYSTICK_CLKSourceConfig()` macro is defined inside the `stm32h7xx_hal_cortex.h` file.
- You can change the SysTick IRQ priority by calling the `HAL_NVIC_SetPriority(SysTick_IRQn,...)` function just after the `HAL_SYSTICK_Config()` function call. The `HAL_NVIC_SetPriority()` call the `NVIC_SetPriority()` function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
 - Reload Value is the parameter to be passed for `HAL_SYSTICK_Config()` function
 - Reload Value should not exceed 0xFFFFFFF

9.2.2 Initialization and de-initialization functions

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

This section contains the following APIs:

- [HAL_NVIC_SetPriorityGrouping](#)
- [HAL_NVIC_SetPriority](#)
- [HAL_NVIC_EnableIRQ](#)
- [HAL_NVIC_DisableIRQ](#)
- [HAL_NVIC_SystemReset](#)
- [HAL_SYSTICK_Config](#)

9.2.3 Peripheral Control functions

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

This section contains the following APIs:

- [HAL_MPU_ConfigRegion](#)
- [HAL_NVIC_GetPriorityGrouping](#)
- [HAL_NVIC_GetPriority](#)
- [HAL_NVIC_SetPendingIRQ](#)
- [HAL_NVIC_GetPendingIRQ](#)
- [HAL_NVIC_ClearPendingIRQ](#)
- [HAL_NVIC_GetActive](#)
- [HAL_SYSTICK_CLKSourceConfig](#)
- [HAL_SYSTICK_IRQHandler](#)
- [HAL_SYSTICK_Callback](#)
- [HAL_GetCurrentCPUID](#)

9.2.4 Detailed description of functions

HAL_NVIC_SetPriorityGrouping

Function name

`void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)`

Function description

Sets the priority grouping field (preemption priority and subpriority) using the required unlock sequence.

Parameters

- **PriorityGroup:** The priority grouping bits length. This parameter can be one of the following values:
 - NVIC_PRIORITYGROUP_0: 0 bits for preemption priority 4 bits for subpriority
 - NVIC_PRIORITYGROUP_1: 1 bits for preemption priority 3 bits for subpriority
 - NVIC_PRIORITYGROUP_2: 2 bits for preemption priority 2 bits for subpriority
 - NVIC_PRIORITYGROUP_3: 3 bits for preemption priority 1 bits for subpriority
 - NVIC_PRIORITYGROUP_4: 4 bits for preemption priority 0 bits for subpriority

Return values

- **None:**

Notes

- When the NVIC_PriorityGroup_0 is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the subpriority.

HAL_NVIC_SetPriority

Function name

```
void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)
```

Function description

Sets the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))
- **PreemptPriority:** The preemption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority
- **SubPriority:** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.

Return values

- **None:**

HAL_NVIC_EnableIRQ

Function name

```
void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)
```

Function description

Enables a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **None:**

Notes

- To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

HAL_NVIC_DisableIRQ

Function name

```
void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)
```

Function description

Disables a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **None:**

HAL_NVIC_SystemReset

Function name

```
void HAL_NVIC_SystemReset (void )
```

Function description

Initiates a system reset request to reset the MCU.

Return values

- **None:**

HAL_SYSTICK_Config

Function name

```
uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)
```

Function description

Initializes the System Timer and its interrupt, and starts the System Tick Timer.

Parameters

- **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.

Return values

- **status:** - 0 Function succeeded.
 - 1 Function failed.

HAL MPU_ConfigRegion

Function name

```
void HAL MPU_ConfigRegion (MPU_Region_InitTypeDef * MPU_Init)
```

Function description

Initializes and configures the Region and the memory to be protected.

Parameters

- **MPU_Init:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

HAL_NVIC_GetPriorityGrouping

Function name

```
uint32_t HAL_NVIC_GetPriorityGrouping (void )
```

Function description

Gets the priority grouping field from the NVIC Interrupt Controller.

Return values

- **Priority:** grouping field (SCB->AIRCR [10:8] PRIGROUP field)

HAL_NVIC_GetPriority

Function name

```
void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority,  
uint32_t * pSubPriority)
```

Function description

Gets the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))
- **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:
 - NVIC_PRIORITYGROUP_0: 0 bits for preemption priority 4 bits for subpriority
 - NVIC_PRIORITYGROUP_1: 1 bits for preemption priority 3 bits for subpriority
 - NVIC_PRIORITYGROUP_2: 2 bits for preemption priority 2 bits for subpriority
 - NVIC_PRIORITYGROUP_3: 3 bits for preemption priority 1 bits for subpriority
 - NVIC_PRIORITYGROUP_4: 4 bits for preemption priority 0 bits for subpriority
- **pPreemptPriority:** Pointer on the Preemptive priority value (starting from 0).
- **pSubPriority:** Pointer on the Subpriority value (starting from 0).

Return values

- **None:**

HAL_NVIC_GetPendingIRQ

Function name

```
uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)
```

Function description

Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
 - 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name

```
void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)
```

Function description

Sets Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **None:**

HAL_NVIC_ClearPendingIRQ

Function name

void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)

Function description

Clears the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **None:**

HAL_NVIC_GetActive

Function name

uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)

Function description

Gets active interrupt (reads the active register in NVIC and returns the active bit).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32h7xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
 - 1 Interrupt status is pending.

HAL_SYSTICK_CLKSourceConfig

Function name

void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)

Function description

Configures the SysTick clock source.

Parameters

- **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values:
 - SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.
 - SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.

Return values

- **None:**

HAL_SYSTICK_IRQHandler

Function name

void HAL_SYSTICK_IRQHandler (void)

Function description

This function handles SYSTICK interrupt request.

Return values

- **None:**

HAL_SYSTICK_Callback

Function name

void HAL_SYSTICK_Callback (void)

Function description

SYSTICK callback.

Return values

- **None:**

HAL_GetCurrentCPUID

Function name

uint32_t HAL_GetCurrentCPUID (void)

Function description

Returns the current CPU ID.

Return values

- **CPU:** identifier

HAL MPU_Disable

Function name

__STATIC_INLINE void HAL MPU_Disable (void)

Function description

Disables the MPU.

Return values

- **None:**

HAL MPU_Enable

Function name

__STATIC_INLINE void HAL MPU_Enable (uint32_t MPU_Control)

Function description

Enables the MPU.

Parameters

- **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values:
 - MPU_HFNMI_PRIVDEF_NONE
 - MPU_HARDFAULT_NMI
 - MPU_PRIVILEGED_DEFAULT
 - MPU_HFNMI_PRIVDEF

Return values

- None:

9.3 CORTEX Firmware driver defines

9.3.1 CORTEX

CORTEX_CPU_Identifier

`CM7_CPUID`

CORTEX MPU Instruction Access Bufferable

`MPU_ACCESS_BUFFERABLE`

`MPU_ACCESS_NOT_BUFFERABLE`

CORTEX MPU Instruction Access Cacheable

`MPU_ACCESS_CACHEABLE`

`MPU_ACCESS_NOT_CACHEABLE`

CORTEX MPU Instruction Access Shareable

`MPU_ACCESS_SHAREABLE`

`MPU_ACCESS_NOT_SHAREABLE`

MPU_HFNMI and PRIVILEGED Access control

`MPU_HFNMI_PRIVDEF_NONE`

`MPU_HARDFAULT_NMI`

`MPU_PRIVILEGED_DEFAULT`

`MPU_HFNMI_PRIVDEF`

CORTEX MPU Instruction Access

`MPU_INSTRUCTION_ACCESS_ENABLE`

`MPU_INSTRUCTION_ACCESS_DISABLE`

CORTEX MPU Region Enable

`MPU_REGION_ENABLE`

`MPU_REGION_DISABLE`

CORTEX MPU Region Number

`MPU_REGION_NUMBER0`

`MPU_REGION_NUMBER1`

`MPU_REGION_NUMBER2`

`MPU_REGION_NUMBER3`

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

CORTEX MPU Region Permission Attributes

MPU_REGION_NO_ACCESS

MPU_REGION_PRIV_RW

MPU_REGION_PRIV_RW_URO

MPU_REGION_FULL_ACCESS

MPU_REGION_PRIV_RO

MPU_REGION_PRIV_RO_URO

CORTEX MPU Region Size

MPU_REGION_SIZE_32B

MPU_REGION_SIZE_64B

MPU_REGION_SIZE_128B

MPU_REGION_SIZE_256B

MPU_REGION_SIZE_512B

MPU_REGION_SIZE_1KB

MPU_REGION_SIZE_2KB

MPU_REGION_SIZE_4KB

MPU_REGION_SIZE_8KB

MPU_REGION_SIZE_16KB

MPU_REGION_SIZE_32KB

MPU_REGION_SIZE_64KB

MPU_REGION_SIZE_128KB

MPU_REGION_SIZE_256KB

MPU_REGION_SIZE_512KB

MPU_REGION_SIZE_1MB

MPU_REGION_SIZE_2MB

`MPU_REGION_SIZE_4MB`

`MPU_REGION_SIZE_8MB`

`MPU_REGION_SIZE_16MB`

`MPU_REGION_SIZE_32MB`

`MPU_REGION_SIZE_64MB`

`MPU_REGION_SIZE_128MB`

`MPU_REGION_SIZE_256MB`

`MPU_REGION_SIZE_512MB`

`MPU_REGION_SIZE_1GB`

`MPU_REGION_SIZE_2GB`

`MPU_REGION_SIZE_4GB`

MPU TEX Levels

`MPU_TEX_LEVEL0`

`MPU_TEX_LEVEL1`

`MPU_TEX_LEVEL2`

CORTEX Preemption Priority Group

`NVIC_PRIORITYGROUP_0`

0 bits for pre-emption priority 4 bits for subpriority

`NVIC_PRIORITYGROUP_1`

1 bits for pre-emption priority 3 bits for subpriority

`NVIC_PRIORITYGROUP_2`

2 bits for pre-emption priority 2 bits for subpriority

`NVIC_PRIORITYGROUP_3`

3 bits for pre-emption priority 1 bits for subpriority

`NVIC_PRIORITYGROUP_4`

4 bits for pre-emption priority 0 bits for subpriority

CORTEX _SysTick clock source

`SYSTICK_CLKSOURCE_HCLK_DIV8`

`SYSTICK_CLKSOURCE_HCLK`

10 HAL CRC Generic Driver

10.1 CRC Firmware driver registers structures

10.1.1 CRC_InitTypeDef

Data Fields

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

Field Documentation

- `uint8_t CRC_InitTypeDef::DefaultPolynomialUse`

This parameter is a value of **Indicates whether or not default polynomial is used** and indicates if default polynomial is used. If set to DEFAULT_POLYNOMIAL_ENABLE, resort to default $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$. In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT_POLYNOMIAL_DISABLE, GeneratingPolynomial and CRCLength fields must be set

- `uint8_t CRC_InitTypeDef::DefaultInitValueUse`

This parameter is a value of **Indicates whether or not default init value is used** and indicates if default init value is used. If set to DEFAULT_INIT_VALUE_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT_INIT_VALUE_DISABLE, InitValue field must be set

- `uint32_t CRC_InitTypeDef::GeneratingPolynomial`

Set CRC generating polynomial. 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal representation, e.g., for a polynomial of degree 7, $X^7 + X^6 + X^5 + X^2 + 1$ is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT_POLYNOMIAL_ENABLE

- `uint32_t CRC_InitTypeDef::CRCLength`

This parameter is a value of **Polynomial sizes to configure the IP** and indicates CRC length. Value can be either one of CRC_POLYLENGTH_32B (32-bit CRC) CRC_POLYLENGTH_16B (16-bit CRC) CRC_POLYLENGTH_8B (8-bit CRC) CRC_POLYLENGTH_7B (7-bit CRC)

- `uint32_t CRC_InitTypeDef::InitValue`

Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE

- `uint32_t CRC_InitTypeDef::InputDataInversionMode`

This parameter is a value of **CRC Extended input data inversion modes** and specifies input data inversion mode. Can be either one of the following values CRC_INPUTDATA_INVERSION_NONE no input data inversion CRC_INPUTDATA_INVERSION_BYTE byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2 CRC_INPUTDATA_INVERSION_HALFWORD halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C CRC_INPUTDATA_INVERSION_WORD word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- `uint32_t CRC_InitTypeDef::OutputDataInversionMode`

This parameter is a value of **CRC Extended output data inversion modes** and specifies output data (i.e. CRC) inversion mode. Can be either CRC_OUTPUTDATA_INVERSION_DISABLE no CRC inversion, or CRC_OUTPUTDATA_INVERSION_ENABLE CRC 0x11223344 is converted into 0x22CC4488

10.1.2 CRC_HandleTypeDef

Data Fields

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

Field Documentation

- **CRC_TypeDef* CRC_HandleTypeDef::Instance**
Register base address
- **CRC_InitTypeDef CRC_HandleTypeDef::Init**
CRC configuration parameters
- **HAL_LockTypeDef CRC_HandleTypeDef::Lock**
CRC Locking object
- **__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State**
CRC communication state
- **uint32_t CRC_HandleTypeDef::InputDataFormat**

This parameter is a value of **CRC input buffer format** and specifies input data format. Can be either
CRC_INPUTDATA_FORMAT_BYTES input data is a stream of bytes (8-bit data)
CRC_INPUTDATA_FORMAT_HALFWORDS input data is a stream of half-words (16-bit data)
CRC_INPUTDATA_FORMAT_WORDS input data is a stream of words (32-bits data) Note that constant
CRC_INPUT_FORMAT_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is
not one of the three values listed above

10.2 CRC Firmware driver API description

10.2.1 CRC How to use this driver

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

10.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the `CRC_InitTypeDef` and create the associated handle

- DeInitialize the CRC peripheral
- Initialize the CRC MSP
- DeInitialize CRC MSP

This section contains the following APIs:

- [HAL_CRC_Init](#)
- [HAL_CRC_DelInit](#)
- [HAL_CRC_MspInit](#)
- [HAL_CRC_MspDelInit](#)

10.2.3 Peripheral Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [HAL_CRC_Accumulate](#)
- [HAL_CRC_Calculate](#)

10.2.4 Peripheral State functions

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

This section contains the following APIs:

- [HAL_CRC_GetState](#)

10.2.5 Detailed description of functions

HAL_CRC_Init

Function name

`HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)`

Function description

Initialize the CRC according to the specified parameters in the `CRC_InitTypeDef` and create the associated handle.

Parameters

- `hcrc`: CRC handle

Return values

- `HAL`: status

HAL_CRC_DelInit

Function name

`HAL_StatusTypeDef HAL_CRC_DelInit (CRC_HandleTypeDef * hcrc)`

Function description

DeInitialize the CRC peripheral.

Parameters

- `hcrc`: CRC handle

Return values

- **HAL:** status

HAL_CRC_MspInit

Function name

```
void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)
```

Function description

Initialize the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_MspDeInit

Function name

```
void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)
```

Function description

DeInitialize the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_Accumulate

Function name

```
uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)
```

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_Calculate

Function name

```
uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)
```

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_GetState

Function name

```
HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)
```

Function description

Return the CRC state.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** state

10.3 CRC Firmware driver defines

10.3.1 CRC

Default CRC computation initialization value

DEFAULT_CRC_INITVALUE

Indicates whether or not default init value is used

DEFAULT_INIT_VALUE_ENABLE

DEFAULT_INIT_VALUE_DISABLE

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

DEFAULT_POLYNOMIAL_DISABLE

Default CRC generating polynomial**DEFAULT_CRC32_POLY*****CRC Exported Functions*****HAL_CRC_Input_Data_Reverse****HAL_CRC_Output_Data_Reverse*****CRC exported macros*****__HAL_CRC_RESET_HANDLE_STATE****Description:**

- Reset CRC handle state.

Parameters:

- __HANDLE__: CRC handle.

Return value:

- None

__HAL_CRC_DR_RESET**Description:**

- Reset CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None.

__HAL_CRC_INITIALCRCVALUE_CONFIG**Description:**

- Set CRC INIT non-default value.

Parameters:

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

Return value:

- None.

__HAL_CRC_SET_IDR**Description:**

- Stores a 32-bit data in the Independent Data(ID) register.

Parameters:

- __HANDLE__: CRC handle
- __VALUE__: 32-bit value to be stored in the ID register

Return value:

- None

__HAL_CRC_GET_IDR**Description:**

- Returns the 32-bit data stored in the Independent Data(ID) register.

Parameters:

- `__HANDLE__`: CRC handle

Return value:

- 32-bit: value of the ID register

CRC input buffer format`CRC_INPUTDATA_FORMAT_UNDEFINED``CRC_INPUTDATA_FORMAT_BYTES``CRC_INPUTDATA_FORMAT_HALFWORDS``CRC_INPUTDATA_FORMAT_WORDS`***Polynomial sizes to configure the IP***`CRC_POLYLENGTH_32B``CRC_POLYLENGTH_16B``CRC_POLYLENGTH_8B``CRC_POLYLENGTH_7B`***CRC polynomial possible sizes actual definitions***`HAL_CRC_LENGTH_32B``HAL_CRC_LENGTH_16B``HAL_CRC_LENGTH_8B``HAL_CRC_LENGTH_7B`

11 HAL CRC Extension Driver

11.1 CRCEEx Firmware driver API description

11.1.1 CRC specific features

1. Polynomial configuration.
2. Input data reverse mode.
3. Output data reverse mode.

11.1.2 CRC Extended features functions

This subsection provides function allowing to:

- Set CRC polynomial if different from default one.

This section contains the following APIs:

- [*HAL_CRCEEx_Polynomial_Set*](#)
- [*HAL_CRCEEx_Input_Data_Reverse*](#)
- [*HAL_CRCEEx_Output_Data_Reverse*](#)

11.1.3 Detailed description of functions

[*HAL_CRCEEx_Polynomial_Set*](#)

Function name

`HAL_StatusTypeDef HAL_CRCEEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)`

Function description

Initializes the CRC polynomial if different from default one.

Parameters

- **hcrc:** CRC handle
- **Pol:** CRC generating polynomial (7, 8, 16 or 32-bit long) This parameter is written in normal representation, e.g. for a polynomial of degree 7, $X^7 + X^6 + X^5 + X^2 + 1$ is written 0x65 for a polynomial of degree 16, $X^{16} + X^{12} + X^5 + 1$ is written 0x1021
- **PolyLength:** CRC polynomial length This parameter can be one of the following values:
 - `CRC_POLYLENGTH_7B`: 7-bit long CRC (generating polynomial of degree 7)
 - `CRC_POLYLENGTH_8B`: 8-bit long CRC (generating polynomial of degree 8)
 - `CRC_POLYLENGTH_16B`: 16-bit long CRC (generating polynomial of degree 16)
 - `CRC_POLYLENGTH_32B`: 32-bit long CRC (generating polynomial of degree 32)

Return values

- **HAL:** status

[*HAL_CRCEEx_Input_Data_Reverse*](#)

Function name

`HAL_StatusTypeDef HAL_CRCEEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)`

Function description

Set the Reverse Input data mode.

Parameters

- **hcrc:** CRC handle
- **InputReverseMode:** Input Data inversion mode This parameter can be one of the following values:
 - CRC_INPUTDATA_INVERSION_NONE: no change in bit order (default value)
 - CRC_INPUTDATA_INVERSION_BYTE: Byte-wise bit reversal
 - CRC_INPUTDATA_INVERSION_HALFWORD: HalfWord-wise bit reversal
 - CRC_INPUTDATA_INVERSION_WORD: Word-wise bit reversal

Return values

- **HAL:** status

HAL_CRCEEx_Output_Data_Reverse

Function name

```
HAL_StatusTypeDef HAL_CRCEEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t  
OutputReverseMode)
```

Function description

Set the Reverse Output data mode.

Parameters

- **hcrc:** CRC handle
- **OutputReverseMode:** Output Data inversion mode This parameter can be one of the following values:
 - CRC_OUTPUTDATA_INVERSION_DISABLE: no CRC inversion (default value)
 - CRC_OUTPUTDATA_INVERSION_ENABLE: bit-level inversion (e.g for a 8-bit CRC: 0xB5 becomes 0xAD)

Return values

- **HAL:** status

11.2 CRCEEx Firmware driver defines

11.2.1 CRCEEx

CRC Extended exported macros

_HAL_CRC_OUTPUTREVERSAL_ENABLE

Description:

- Set CRC output reversal.

Parameters:

- _HANDLE_: : CRC handle

Return value:

- None.

_HAL_CRC_OUTPUTREVERSAL_DISABLE

Description:

- Unset CRC output reversal.

Parameters:

- `__HANDLE__`: : CRC handle

Return value:

- None.

`_HAL_CRC_POLYNOMIAL_CONFIG`**Description:**

- Set CRC non-default polynomial.

Parameters:

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

Return value:

- None.

CRC Extended input data inversion modes**`CRC_INPUTDATA_INVERSION_NONE`****`CRC_INPUTDATA_INVERSION_BYTE`****`CRC_INPUTDATA_INVERSION_HALFWORD`****`CRC_INPUTDATA_INVERSION_WORD`****`IS_CRC_INPUTDATA_INVERSION_MODE`*****CRC Extended output data inversion modes*****`CRC_OUTPUTDATA_INVERSION_DISABLE`****`CRC_OUTPUTDATA_INVERSION_ENABLE`****`IS_CRC_OUTPUTDATA_INVERSION_MODE`**

12 HAL CRYP Generic Driver

12.1 CRYP Firmware driver registers structures

12.1.1 CRYP_ConfigTypeDef

Data Fields

- `uint32_t DataType`
- `uint32_t KeySize`
- `uint32_t * pKey`
- `uint32_t * pInitVect`
- `uint32_t Algorithm`
- `uint32_t * Header`
- `uint32_t HeaderSize`
- `uint32_t * B0`

Field Documentation

- `uint32_t CRYP_ConfigTypeDef::DataType`

32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of **CRYP Data Type**

- `uint32_t CRYP_ConfigTypeDef::KeySize`

Used only in AES mode : 128, 192 or 256 bit key length in CRYP1. This parameter can be a value of **CRYP Key Size**

- `uint32_t* CRYP_ConfigTypeDef::pKey`

The key used for encryption/decryption

- `uint32_t* CRYP_ConfigTypeDef::pInitVect`

The initialization vector used also as initialization counter in CTR mode

- `uint32_t CRYP_ConfigTypeDef::Algorithm`

DES/ TDES Algorithm ECB/CBC AES Algorithm ECB/CBC/CTR/GCM or CCM This parameter can be a value of **CRYP Algorithm Mode**

- `uint32_t* CRYP_ConfigTypeDef::Header`

used only in AES GCM and CCM Algorithm for authentication, GCM : also known as Additional Authentication Data CCM : named B1 composed of the associated data length and Associated Data.

- `uint32_t CRYP_ConfigTypeDef::HeaderSize`

The size of header buffer in word

- `uint32_t* CRYP_ConfigTypeDef::B0`

B0 is first authentication block used only in AES CCM mode

12.1.2 CRYP_HandleTypeDef

Data Fields

- `CRYP_TypeDef * Instance`
- `CRYP_ConfigTypeDef Init`
- `uint32_t * pCrypInBuffPtr`
- `uint32_t * pCrypOutBuffPtr`

- `__IO uint16_t CrypHeaderCount`
- `__IO uint16_t CrypInCount`
- `__IO uint16_t CrypOutCount`
- `uint16_t Size`
- `uint32_t Phase`
- `DMA_HandleTypeDef * hdmain`
- `DMA_HandleTypeDef * hdmaout`
- `HAL_LockTypeDef Lock`
- `__IO HAL_CRYP_STATETypeDef State`
- `__IO uint32_t ErrorCode`

Field Documentation

- **`CRYP_TypeDef* CRYP_HandleTypeDef::Instance`**
CRYP registers base address
- **`CRYP_ConfigTypeDef CRYP_HandleTypeDef::Init`**
CRYP required parameters
- **`uint32_t* CRYP_HandleTypeDef::pCrypInBuffPtr`**
Pointer to CRYP processing (encryption, decryption,...) buffer
- **`uint32_t* CRYP_HandleTypeDef::pCrypOutBuffPtr`**
Pointer to CRYP processing (encryption, decryption,...) buffer
- **`__IO uint16_t CRYP_HandleTypeDef::CrypHeaderCount`**
Counter of header data
- **`__IO uint16_t CRYP_HandleTypeDef::CrypInCount`**
Counter of input data
- **`__IO uint16_t CRYP_HandleTypeDef::CrypOutCount`**
Counter of output data
- **`uint16_t CRYP_HandleTypeDef::Size`**
length of input data in word
- **`uint32_t CRYP_HandleTypeDef::Phase`**
CRYP peripheral phase
- **`DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmain`**
CRYP In DMA handle parameters
- **`DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmaout`**
CRYP Out DMA handle parameters
- **`HAL_LockTypeDef CRYP_HandleTypeDef::Lock`**
CRYP locking object
- **`__IO HAL_CRYP_STATETypeDef CRYP_HandleTypeDef::State`**
CRYP peripheral state
- **`__IO uint32_t CRYP_HandleTypeDef::ErrorCode`**
CRYP peripheral error code

12.2

CRYP Firmware driver API description

12.2.1 How to use this driver

The CRYP HAL driver can be used in CRYP IP as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
 - a. Enable the CRYP interface clock using __HAL_RCC_CRYP_CLK_ENABLE()
 - b. In case of using interrupts (e.g. HAL_CRYP_Encrypt_IT())
 - Configure the CRYP interrupt priority using HAL_NVIC_SetPriority()
 - Enable the CRYP IRQ handler using HAL_NVIC_EnableIRQ()
 - In CRYP IRQ handler, call HAL_CRYP_IRQHandler()
 - c. In case of using DMA to control data transfer (e.g. HAL_CRYP_Encrypt_DMA())
 - Enable the DMAx interface clock using __RCC_DMAMUX_CLK_ENABLE()
 - Configure and enable two DMA streams one for managing data transfer from memory to peripheral (input stream) and another stream for managing data transfer from peripheral to memory (output stream)
 - Associate the initialized DMA handle to the CRYP DMA handle using __HAL_LINKDMA()
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the CRYP according to the specified parameters :
 - a. The data type: 1-bit, 8-bit, 16-bit or 32-bit.
 - b. The key size: 128, 192 or 256.
 - c. The AlgoMode DES/ TDES Algorithm ECB/CBC or AES Algorithm ECB/CBC/CTR/GCM or CCM.
 - d. The initialization vector (counter). It is not used in ECB mode.
 - e. The key buffer used for encryption/decryption.
 - f. The Header used only in AES GCM and CCM Algorithm for authentication.
 - g. The HeaderSize The size of header buffer in word.
 - h. The B0 block is the first authentication block used only in AES CCM mode.
3. Three processing (encryption/decryption) functions are available:
 - a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished, e.g. HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
 - b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
 - c. DMA mode: encryption and decryption APIs are not blocking functions i.e. the data transfer is ensured by DMA, e.g. HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA
4. When the processing function is called at first time after HAL_CRYP_Init() the CRYP peripheral is configured and processes the buffer in input. At second call, no need to Initialize the CRYP, user have to get current configuration via HAL_CRYP_GetConfig() API, then only HAL_CRYP_SetConfig() is requested to set new parameters, finally user can start encryption/decryption.
5. Call HAL_CRYP_DeInit() to deinitialize the CRYP peripheral.

The cryptographic processor supports following standards:

1. The data encryption standard (DES) and Triple-DES (TDES) supported only by CRYP1 IP:
 - a. 64-bit data block processing
 - b. chaining modes supported :
 - Electronic Code Book(ECB)
 - Cipher Block Chaining (CBC)
 - c. keys length supported :64-bit, 128-bit and 192-bit.
2. The advanced encryption standard (AES) supported by CRYP1:
 - a. 128-bit data block processing
 - b. chaining modes supported :
 - Electronic Code Book(ECB)
 - Cipher Block Chaining (CBC)

- Counter mode (CTR)
 - Galois/counter mode (GCM/GMAC)
 - Counter with Cipher Block Chaining-Message(CCM)
- c. keys length Supported :
- for CRYP1 IP: 128-bit, 192-bit and 256-bit.

This section describes the AES Galois/counter mode (GCM) supported by both CRYP1 IP:

1. Algorithm supported :
 - a. Galois/counter mode (GCM)
 - b. Galois message authentication code (GMAC) :is exactly the same as GCM algorithm composed only by an header.
2. Four phases are performed in GCM :
 - a. Init phase: IP prepares the GCM hash subkey (H) and do the IV processing
 - b. Header phase: IP processes the Additional Authenticated Data (AAD), with hash computation only.
 - c. Payload phase: IP processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
 - d. Final phase: IP generates the authenticated tag (T) using the last block of data.
3. structure of message construction in GCM is defined as below :
 - a. 16 bytes Initial Counter Block (ICB)composed of IV and counter
 - b. The authenticated header A (also knows as Additional Authentication Data AAD) this part of the message is only authenticated, not encrypted.
 - c. The plaintext message P is both authenticated and encrypted as ciphertext. GCM standard specifies that ciphertext has same bit length as the plaintext.
 - d. The last block is composed of the length of A (on 64 bits) and the length of ciphertext (on 64 bits)

This section describe The AES Counter with Cipher Block Chaining-Message Authentication Code (CCM) supported by both CRYP1 IP:

1. Specific parameters for CCM :
 - a. B0 block : According to NIST Special Publication 800-38C, The first block B0 is formatted as follows, where l(m) is encoded in most-significant-byte first order(see below table 3)
 - Q: a bit string representation of the octet length of P (plaintext)
 - q The octet length of the binary representation of the octet length of the payload
 - A nonce (N), n The octet length of the where n+q=15.
 - Flags: most significant octet containing four flags for control information,
 - t The octet length of the MAC.
 - b. B1 block (header) : associated data length(a) concatenated with Associated Data (A) the associated data length expressed in bytes (a) defined as below:
 - If $0 < a < 216-28$, then it is encoded as [a]16, i.e. two octets
 - If $216-28 < a < 232$, then it is encoded as 0xff || 0xfe || [a]32, i.e. six octets
 - If $232 < a < 264$, then it is encoded as 0xff || 0xff || [a]64, i.e. ten octets
 - c. CTRx block : control blocks
 - Generation of CTR1 from first block B0 information : equal to B0 with first 5 bits zeroed and most significant bits storing octet length of P also zeroed, then incremented by one (see below Table 4)
 - Generation of CTR0: same as CTR1 with bit[0] set to zero.
2. Four phases are performed in CCM for CRYP1 IP:
 - a. Init phase: IP prepares the GCM hash subkey (H) and do the IV processing
 - b. Header phase: IP processes the Additional Authenticated Data (AAD), with hash computation only.
 - c. Payload phase: IP processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
 - d. Final phase: IP generates the authenticated tag (T) using the last block of data.

12.2.2 Initialization, de-initialization and Set and Get configuration functions

This section provides functions allowing to:

- Initialize the CRYP
- Deinitialize the CRYP
- Initialize the CRYP MSP
- Deinitialize the CRYP MSP
- configure CRYP (HAL_CRYP_SetConfig) with the specified parameters in the CRYP_ConfigTypeDef
Parameters which are configured in This section are :
 - Key size
 - Data Type : 32,16, 8 or 1bit
 - AlgoMode : for CRYP1 IP ECB and CBC in DES/TDES Standard ECB,CBC,CTR,GCM/GMAC and CCM in AES Standard.
- Get CRYP configuration (HAL_CRYP_GetConfig) from the specified parameters in the CRYP_HandleTypeDef

This section contains the following APIs:

- [**HAL_CRYP_Init**](#)
- [**HAL_CRYP_DeInit**](#)
- [**HAL_CRYP_SetConfig**](#)
- [**HAL_CRYP_GetConfig**](#)
- [**HAL_CRYP_MspInit**](#)
- [**HAL_CRYP_MspDeInit**](#)

12.2.3 Encrypt Decrypt functions

This section provides API allowing to Encrypt/Decrypt Data following Standard DES/TDES or AES, and Algorithm configured by the user:

- Standard DES/TDES only supported by CRYP1 IP, below list of Algorithm supported :
 - Electronic Code Book(ECB)
 - Cipher Block Chaining (CBC)
- Standard AES supported by CRYP1 IP , list of Algorithm supported:
 - Electronic Code Book(ECB)
 - Cipher Block Chaining (CBC)
 - Counter mode (CTR)
 - Cipher Block Chaining (CBC)
 - Counter mode (CTR)
 - Galois/counter mode (GCM)
 - Counter with Cipher Block Chaining-Message(CCM)

Three processing functions are available:

- Polling mode : HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
- Interrupt mode : HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
- DMA mode : HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA

This section contains the following APIs:

- [**HAL_CRYP_Encrypt**](#)
- [**HAL_CRYP_Decrypt**](#)
- [**HAL_CRYP_Encrypt_IT**](#)
- [**HAL_CRYP_Decrypt_IT**](#)
- [**HAL_CRYP_Encrypt_DMA**](#)
- [**HAL_CRYP_Decrypt_DMA**](#)

12.2.4 CRYP IRQ handler management

This section provides CRYP IRQ handler and callback functions.

- HAL_CRYP_IRQHandler CRYP interrupt request
- HAL_CRYP_InCpltCallback input data transfer complete callback
- HAL_CRYP_OutCpltCallback output data transfer complete callback
- HAL_CRYP_ErrorCallback CRYP error callback
- HAL_CRYP_GetState return the CRYP state
- HAL_CRYP_GetError return the CRYP error code

This section contains the following APIs:

- [HAL_CRYP_IRQHandler](#)
- [HAL_CRYP_GetError](#)
- [HAL_CRYP_GetState](#)
- [HAL_CRYP_InCpltCallback](#)
- [HAL_CRYP_OutCpltCallback](#)
- [HAL_CRYP_ErrorCallback](#)

12.2.5 Detailed description of functions

HAL_CRYP_Init

Function name

`HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef * hcryp)`

Function description

Initializes the CRYP according to the specified parameters in the CRYP_ConfigTypeDef and creates the associated handle.

Parameters

- **hcryp**: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL**: status

HAL_CRYP_DelInit

Function name

`HAL_StatusTypeDef HAL_CRYP_DelInit (CRYP_HandleTypeDef * hcryp)`

Function description

De-Initializes the CRYP peripheral.

Parameters

- **hcryp**: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL**: status

HAL_CRYP_MspInit

Function name

`void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)`

Function description

Initializes the CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_MspInit

Function name

void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)

Function description

Deinitializes CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_SetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_SetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Configure the CRYP according to the specified parameters in the CRYP_ConfigTypeDef.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure
- **pConf:** pointer to a CRYP_ConfigTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_GetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_GetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Get CRYP Configuration parameters in associated handle.

Parameters

- **pConf:** pointer to a CRYP_ConfigTypeDef structure
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_Encrypt

Function name

```
HAL_StatusTypeDef HAL_CRYP_Encrypt (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size,  
uint32_t * Output, uint32_t Timeout)
```

Function description

Encryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(ciphertext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Decrypt

Function name

```
HAL_StatusTypeDef HAL_CRYP_Decrypt (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t Size,  
uint32_t * Output, uint32_t Timeout)
```

Function description

Decryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(plaintext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Encrypt_IT

Function name

```
HAL_StatusTypeDef HAL_CRYP_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t  
Size, uint32_t * Output)
```

Function description

Encryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)

- **Size:** Length of the plaintext buffer in word
- **Output:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_IT

Function name

```
HAL_StatusTypeDef HAL_CRYP_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t  
Size, uint32_t * Output)
```

Function description

Decryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_Encrypt_DMA

Function name

```
HAL_StatusTypeDef HAL_CRYP_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t  
Size, uint32_t * Output)
```

Function description

Encryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **Input:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer in word.
- **Output:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_DMA

Function name

```
HAL_StatusTypeDef HAL_CRYP_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * Input, uint16_t  
Size, uint32_t * Output)
```

Function description

Decryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

- **Input:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the plaintext buffer in word
- **Output:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_IRQHandler

Function name

```
void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)
```

Function description

This function handles cryptographic interrupt request.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_GetState

Function name

```
HAL_CRYP_STATETypeDef HAL_CRYP_GetState (CRYP_HandleTypeDef * hcryp)
```

Function description

Returns the CRYP state.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **HAL:** state

HAL_CRYP_InCpltCallback

Function name

```
void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)
```

Function description

Input FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_OutCpltCallback

Function name

```
void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)
```

Function description

Output FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_ErrorCallback

Function name

void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)

Function description

CRYP error callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_GetError

Function name

uint32_t HAL_CRYP_GetError (CRYP_HandleTypeDef * hcryp)

Function description

Return the CRYP error code.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for the CRYP IP

Return values

- **CRYP:** error code

12.3 CRYP Firmware driver defines

12.3.1 CRYP

CRYP Algorithm Mode

CRYP_DES_ECB

CRYP_DES_CBC

CRYP_TDES_ECB

CRYP_TDES_CBC

CRYP_AES_ECB

`CRYP_AES_CBC`

`CRYP_AES_CTR`

`CRYP_AES_GCM`

`CRYP_AES_CCM`

CRYP Data Type

`CRYP_DATATYPE_32B`

`CRYP_DATATYPE_16B`

`CRYP_DATATYPE_8B`

`CRYP_DATATYPE_1B`

CRYP Error Definition

`HAL_CRYP_ERROR_NONE`

No error

`HAL_CRYP_ERROR_WRITE`

Write error

`HAL_CRYP_ERROR_READ`

Read error

`HAL_CRYP_ERROR_DMA`

DMA error

`HAL_CRYP_ERROR_BUSY`

Busy flag error

`HAL_CRYP_ERROR_TIMEOUT`

Timeout error

`HAL_CRYP_ERROR_NOT_SUPPORTED`

Not supported mode

`HAL_CRYP_ERROR_AUTH_TAG_SEQUENCE`

Sequence are not respected only for GCM or CCM

CRYP Exported Macros

`_HAL_CRYP_ENABLE`

Description:

- Enable/Disable the CRYP peripheral.

Parameters:

- `_HANDLE_`: specifies the CRYP handle.

Return value:

- None

`_HAL_CRYP_DISABLE`

CRYP_FLAG_MASK

Description:

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

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values for CRYP:
 - CRYP_FLAG_BUSY: The CRYP core is currently processing a block of data or a key preparation (for AES decryption).
 - CRYP_FLAG_IFEM: Input FIFO is empty
 - CRYP_FLAG_INNF: Input FIFO is not full
 - CRYP_FLAG_INRIS: Input FIFO service raw interrupt is pending
 - CRYP_FLAG_OFNE: Output FIFO is not empty
 - CRYP_FLAG_OFFU: Output FIFO is full
 - CRYP_FLAG_OUTRIS: Input FIFO service raw interrupt is pending

Return value:

- The state of __FLAG__ (TRUE or FALSE).

__HAL_CRYP_GET_FLAG

__HAL_CRYP_GET_IT

Description:

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

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: specifies the interrupt to check. This parameter can be one of the following values for CRYP:
 - CRYP_IT_INI: Input FIFO service masked interrupt status
 - CRYP_IT_OUTI: Output FIFO service masked interrupt status

Return value:

- The state of __INTERRUPT__ (TRUE or FALSE).

__HAL_CRYP_ENABLE_IT

Description:

- Enable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for CRYP: @ CRYP_IT_INI : Input FIFO service interrupt mask. @ CRYP_IT_OUTI : Output FIFO service interrupt mask.CRYP interrupt.

Return value:

- None

__HAL_CRYP_DISABLE_IT

Description:

- Disable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.

- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for CRYP: @ CRYP_IT_INI : Input FIFO service interrupt mask. @ CRYP_IT_OUTI : Output FIFO service interrupt mask.CRYP interrupt.

Return value:

- None

CRYP Flags**CRYP_FLAG_IFEM**

Input FIFO is empty

CRYP_FLAG_IFNF

Input FIFO is not Full

CRYP_FLAG_OFNE

Output FIFO is not empty

CRYP_FLAG_OFFU

Output FIFO is Full

CRYP_FLAG_BUSY

The CRYP core is currently processing a block of data or a key preparation (for AES decryption).

CRYP_FLAG_OUTRIS

Output FIFO service raw interrupt status

CRYP_FLAG_INRIS

Input FIFO service raw interrupt status

CRYP Interrupt**CRYP_IT_INI**

Input FIFO Interrupt

CRYP_IT_OUTI

Output FIFO Interrupt

CRYP Private macros to check input parameters**IS_CRYP_ALGORITHM****IS_CRYP_KEYSIZE****IS_CRYP_DATATYPE****CRYP Key Size****CRYP_KEYSIZE_128B****CRYP_KEYSIZE_192B****CRYP_KEYSIZE_256B**

13 HAL CRYP Extension Driver

13.1 CRYPEx Firmware driver API description

13.1.1 How to use this driver

The CRYP extension HAL driver can be used after AES-GCM or AES-CCM Encryption/Decryption to get the authentication messages.

13.1.2 Extended AES processing functions

This section provides functions allowing to generate the authentication TAG in Polling mode

- `HAL_CRYPEx_AESGCM_GenerateAuthTAG`
- `HAL_CRYPEx_AESCCM_GenerateAuthTAG` they should be used after Encrypt/Decrypt operation.

This section contains the following APIs:

- `HAL_CRYPEx_AESGCM_GenerateAuthTAG`
- `HAL_CRYPEx_AESCCM_GenerateAuthTAG`

13.1.3 Detailed description of functions

`HAL_CRYPEx_AESGCM_GenerateAuthTAG`

Function name

`HAL_StatusTypeDef HAL_CRYPEx_AESGCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, uint32_t * AuthTag, uint32_t Timeout)`

Function description

generate the GCM authentication TAG.

Parameters

- **hcryp**: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **AuthTag**: Pointer to the authentication buffer
- **Timeout**: Timeout duration

Return values

- **HAL**: status

`HAL_CRYPEx_AESCCM_GenerateAuthTAG`

Function name

`HAL_StatusTypeDef HAL_CRYPEx_AESCCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, uint32_t * AuthTag, uint32_t Timeout)`

Function description

AES CCM Authentication TAG generation.

Parameters

- **hcryp**: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **AuthTag**: Pointer to the authentication buffer
- **Timeout**: Timeout duration

Return values

- **HAL:** status

14 HAL DAC Generic Driver

14.1 DAC Firmware driver registers structures

14.1.1 DAC_HandleTypeDef

Data Fields

- `DAC_TypeDef * Instance`
- `__IO HAL_DAC_StateTypeDef State`
- `HAL_LockTypeDef Lock`
- `DMA_HandleTypeDef * DMA_Handle1`
- `DMA_HandleTypeDef * DMA_Handle2`
- `__IO uint32_t ErrorCode`

Field Documentation

- **`DAC_TypeDef* DAC_HandleTypeDef::Instance`**
Register base address
- **`__IO HAL_DAC_StateTypeDef DAC_HandleTypeDef::State`**
DAC communication state
- **`HAL_LockTypeDef DAC_HandleTypeDef::Lock`**
DAC locking object
- **`DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle1`**
Pointer DMA handler for channel 1
- **`DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle2`**
Pointer DMA handler for channel 2
- **`__IO uint32_t DAC_HandleTypeDef::ErrorCode`**
DAC Error code

14.1.2 DAC_SampleAndHoldConfTypeDef

Data Fields

- `uint32_t DAC_SampleTime`
- `uint32_t DAC_HoldTime`
- `uint32_t DAC_RefreshTime`

Field Documentation

- **`uint32_t DAC_SampleAndHoldConfTypeDef::DAC_SampleTime`**
Specifies the Sample time for the selected channel. This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- **`uint32_t DAC_SampleAndHoldConfTypeDef::DAC_HoldTime`**
Specifies the hold time for the selected channel. This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- **`uint32_t DAC_SampleAndHoldConfTypeDef::DAC_RefreshTime`**

Specifies the refresh time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 255

14.1.3 DAC_ChannelConfTypeDef

Data Fields

- `uint32_t DAC_SampleAndHold`
- `uint32_t DAC_Trigger`
- `uint32_t DAC_OutputBuffer`
- `uint32_t DAC_ConnectOnChipPeripheral`
- `uint32_t DAC_UserTrimming`
- `uint32_t DAC_TrimmingValue`
- `DAC_SampleAndHoldConfTypeDef DAC_SampleAndHoldConfig`

Field Documentation

- **`uint32_t DAC_ChannelConfTypeDef::DAC_SampleAndHold`**
Specifies whether the DAC mode. This parameter can be a value of . **Mode is Sample and hold (low power or normal)**
- **`uint32_t DAC_ChannelConfTypeDef::DAC_Trigger`**
Specifies the external trigger for the selected DAC channel. This parameter can be a value of **DAC trigger selection**
- **`uint32_t DAC_ChannelConfTypeDef::DAC_OutputBuffer`**
Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of **DAC output buffer**
- **`uint32_t DAC_ChannelConfTypeDef::DAC_ConnectOnChipPeripheral`**
Specifies whether the DAC output is connected or not to on chip peripheral . This parameter can be a value of **DAC ConnectOnChipPeripheral**
- **`uint32_t DAC_ChannelConfTypeDef::DAC_UserTrimming`**
Specifies the trimming mode This parameter must be a value of **DAC User Trimming** DAC_UserTrimming is either factory or user trimming
- **`uint32_t DAC_ChannelConfTypeDef::DAC_TrimmingValue`**
Specifies the offset trimming value i.e. when DAC_SampleAndHold is DAC_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- **`DAC_SampleAndHoldConfTypeDef DAC_ChannelConfTypeDef::DAC_SampleAndHoldConfig`**
Sample and Hold settings

14.2 DAC Firmware driver API description

14.2.1 DAC Peripheral features

DAC Channels

STM32H7 devices integrate two 12-bit Digital Analog Converters. The 2 converters (i.e. channel1 & channel2) can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 (PA4) as output or connected to on-chip peripherals (ex. OPAMPS, comparators).
2. DAC channel2 with DAC_OUT2 (PA5) as output or connected to on-chip peripherals (ex. OPAMPS, comparators).

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO:TIM1,TIM2,TIM4, TIM5, TIM6, TIM7,TIM8 and TIM15 (DAC_TRIGGER_T1_TRGO, DAC_TRIGGER_T2_TRGO...)
3. Timers TRGO: HRTIM1,LPTIM1,LPTIM2 (DAC_TRIGGER_HR1_TRGO1,DAC_TRIGGER_HR1_TRGO2,DAC_TRIGGER_LP1_OUT,DAC_TRIGGER_LP2_OUT)
4. Software using DAC_TRIGGER_SOFTWARE

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;

Note:

Refer to the device datasheet for more details about output impedance value with and without output buffer.

DAC connect feature

Each DAC channel can be connected internally. To connect, use sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_ENABLE;

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel2 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

DAC Sample and Hold feature

For each converter, 2 modes are supported: normal mode and "sample and hold" mode (i.e. low power mode). In the sample and hold mode, the DAC core converts data, then holds the converted voltage on a capacitor. When not converting, the DAC cores and buffer are completely turned off between samples and the DAC output is tri-stated, therefore reducing the overall power consumption. A new stabilization period is needed before each new conversion.

The sample and hold allow setting internal or external voltage @ low power consumption cost (output value can be at any given rate either by CPU or DMA).

The Sample and hold block and registers uses either LSI & run in several power modes: run mode, sleep mode & stop mode. To enable Sample and Hold mode ,enable LSI using HAL_RCC_OscConfig with RCC_OSCILLATORTYPE_LSI & RCC_LSI_ON parameters. Use DAC_InitStructure.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_ENABLE & DAC_ChannelConfTypeDef.DAC_SampleAndHoldConfig.DAC_SampleTime, DAC_HoldTime & DAC_RefershTime.

DAC calibration feature

1. The 2 converters (channel1 & channel2) provide calibration capabilities.
 - Calibration aims at correcting some offset of output buffer.
 - The DAC uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).

- The user defined settings can be figured out using self calibration handled by HAL_DACEx_SelfCalibrate.
- HAL_DACEx_SelfCalibrate:
 - Runs automatically the calibration.
 - Enables the user trimming mode
 - Updates a structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)

DAC wave generation feature

Both DAC channels can be used to generate:

1. Noise wave
2. Triangle wave

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation:

$$\text{DAC_OUTx} = \text{VREF+} * \text{DOR} / 4095$$

- with DOR is the Data Output Register

VREF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V:

- Assuming that VREF+ = 3.3V, DAC_OUT1 = $(3.3 * 868) / 4095 = 0.7V$

DMA requests

A DMA request can be generated when an external trigger (but not a software trigger) occurs if DMA requests are enabled using HAL_DAC_Start_DMA(). DMA requests are mapped as following:

1. DAC channel1: mapped on DMA_REQUEST_DAC1
 2. DAC channel2: mapped on DMA_REQUEST_DAC2
- @- For Dual mode and specific signal (Triangle and noise) generation please refer to Extended Features Driver description

14.2.2

How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA() functions.

Calibration mode IO operation

- Retrieve the factory trimming (calibration settings) using HAL_DACEx_GetTrimOffset()
- Run the calibration using HAL_DACEx_SelfCalibrate()
- Update the trimming while DAC running using HAL_DACEx_SetUserTrimming()

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion.
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACE_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACE_ConvHalfCpltCallbackCh2().
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACE_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACE_ConvHalfCpltCallbackCh2().
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1().
- In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACE_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACE_DMAUnderrunCallbackCh2() and add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1().
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

DAC HAL driver macros list

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

- `__HAL_DAC_ENABLE` : Enable the DAC peripheral
- `__HAL_DAC_DISABLE` : Disable the DAC peripheral
- `__HAL_DAC_CLEAR_FLAG`: Clear the DAC's pending flags
- `__HAL_DAC_GET_FLAG`: Get the selected DAC's flag status

Note:

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

14.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [`HAL_DAC_Init`](#)
- [`HAL_DAC_DeInit`](#)
- [`HAL_DAC_MspInit`](#)
- [`HAL_DAC_MspDeInit`](#)

14.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.

- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- [HAL_DAC_Start](#)
- [HAL_DAC_Stop](#)
- [HAL_DAC_Start_DMA](#)
- [HAL_DAC_Stop_DMA](#)
- [HAL_DAC_IRQHandler](#)
- [HAL_DAC_SetValue](#)
- [HAL_DAC_ConvCpltCallbackCh1](#)
- [HAL_DAC_ConvHalfCpltCallbackCh1](#)
- [HAL_DAC_ErrorCallbackCh1](#)
- [HAL_DAC_DMADebugCallbackCh1](#)

14.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Get result of conversion.

This section contains the following APIs:

- [HAL_DAC_GetValue](#)
- [HAL_DAC_ConfigChannel](#)

14.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:

- [HAL_DAC_GetState](#)
- [HAL_DAC_GetError](#)

14.2.7 Detailed description of functions

HAL_DAC_Init

Function name

`HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)`

Function description

Initialize the DAC peripheral according to the specified parameters in the DAC_InitStruct and initialize the associated handle.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_DelInit

Function name

```
HAL_StatusTypeDef HAL_DAC_DelInit (DAC_HandleTypeDef * hdac)
```

Function description

Deinitialize the DAC peripheral registers to their default reset values.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_MspInit

Function name

```
void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)
```

Function description

Initialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_MspDelInit

Function name

```
void HAL_DAC_MspDelInit (DAC_HandleTypeDef * hdac)
```

Function description

Deinitialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_Start

Function name

```
HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)
```

Function description

Enable DAC and start conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Stop

Function name

HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disable DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Start_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enable DAC and start conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The destination peripheral Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
 - DAC_ALIGN_8B_R: 8bit right data alignment selected
 - DAC_ALIGN_12B_L: 12bit left data alignment selected
 - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DAC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disable DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_IRQHandler

Function name

```
void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)
```

Function description

Handle DAC interrupt request This function uses the interruption of DMA underrun.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_SetValue

Function name

```
HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)
```

Function description

Set the specified data holding register value for DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected
- **Alignment:** Specifies the data alignment. This parameter can be one of the following values:
 - DAC_ALIGN_8B_R: 8bit right data alignment selected
 - DAC_ALIGN_12B_L: 12bit left data alignment selected
 - DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data:** Data to be loaded in the selected data holding register.

Return values

- **HAL:** status

HAL_DAC_ConvCpltCallbackCh1

Function name

```
void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)
```

Function description

Conversion complete callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name

```
void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)
```

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ErrorCallbackCh1

Function name

```
void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)
```

Function description

Error DAC callback for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_DMADebugCallbackCh1

Function name

```
void HAL_DAC_DMADebugCallbackCh1 (DAC_HandleTypeDef * hdac)
```

Function description

DMA debug DAC callback for channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Configure the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_GetValue

Function name

uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Return the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **The:** selected DAC channel data output value.

HAL_DAC_GetState

Function name

HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)

Function description

return the DAC handle state

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** state

HAL_DAC_GetError**Function name**

`uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)`

Function description

Return the DAC error code.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **DAC:** Error Code

14.3 DAC Firmware driver defines

14.3.1 DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC ConnectOnChipPeripheral

DAC_CHIPCONNECT_DISABLE

DAC_CHIPCONNECT_ENABLE

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE

No error

HAL_DAC_ERROR_DMAUNDERUNCH1

DAC channel1 DMA underrun error

HAL_DAC_ERROR_DMAUNDERUNCH2

DAC channel2 DMA underrun error

HAL_DAC_ERROR_DMA

DMA error

HAL_DAC_ERROR_TIMEOUT

Timeout error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE

Description:

- Reset DAC handle state.

Parameters:

- __HANDLE__: specifies the DAC handle.

Return value:

- None

__HAL_DAC_ENABLE

Description:

- Enable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __DAC_Channel__: specifies the DAC channel

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle
- __DAC_Channel__: specifies the DAC channel.

Return value:

- None

DAC_DHR12R1_ALIGNMENT

Description:

- Set DHR12R1 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12R2_ALIGNMENT

Description:

- Set DHR12R2 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12RD_ALIGNMENT

Description:

- Set DHR12RD alignment.

Parameters:

- `__ALIGNMENT__`: specifies the DAC alignment

Return value:

- None

[__HAL_DAC_ENABLE_IT](#)**Description:**

- Enable the DAC interrupt.

Parameters:

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt. This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
 - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

Return value:

- None

[__HAL_DAC_DISABLE_IT](#)**Description:**

- Disable the DAC interrupt.

Parameters:

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt. This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
 - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

Return value:

- None

[__HAL_DAC_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- `__HANDLE__`: DAC handle
- `__INTERRUPT__`: DAC interrupt source to check. This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
 - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

Return value:

- State: of interruption (SET or RESET)

[__HAL_DAC_GET_FLAG](#)**Description:**

- Get the selected DAC's flag status.

Parameters:

- `__HANDLE__`: specifies the DAC handle.
- `__FLAG__`: specifies the DAC flag to get. This parameter can be any combination of the following values:

- DAC_FLAG_DMAUDR1: DAC channel 1 DMA underrun flag
- DAC_FLAG_DMAUDR2: DAC channel 2 DMA underrun flag

Return value:

- None

_HAL_DAC_CLEAR_FLAG**Description:**

- Clear the DAC's flag.

Parameters:

- `__HANDLE__`: specifies the DAC handle.
- `__FLAG__`: specifies the DAC flag to clear. This parameter can be any combination of the following values:
 - DAC_FLAG_DMAUDR1: DAC channel 1 DMA underrun flag
 - DAC_FLAG_DMAUDR2: DAC channel 2 DMA underrun flag

Return value:

- None

DAC flags definition**DAC_FLAG_DMAUDR1****DAC_FLAG_DMAUDR2****DAC IT definition****DAC_IT_DMAUDR1****DAC_IT_DMAUDR2****DAC output buffer****DAC_OUTPUTBUFFER_ENABLE****DAC_OUTPUTBUFFER_DISABLE****. Mode is Sample and hold (low power or normal)****DAC_SAMPLEANDHOLD_DISABLE****DAC_SAMPLEANDHOLD_ENABLE****DAC trigger selection****DAC_TRIGGER_NONE**

Conversion is automatic once the DAC_DHRxxxx register has been loaded, and not by external trigger

DAC_TRIGGER_SOFTWARE

Conversion started by software trigger for DAC channel

DAC_TRIGGER_T1_TRGO

TIM1 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T2_TRGO

TIM2 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T4_TRGO

TIM4 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T5_TRGO

TIM5 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T6_TRGO

TIM6 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T7_TRGO

TIM7 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T8_TRGO

TIM8 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T15_TRGO

TIM15 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_HR1_TRGO1

HR1 TRGO1 selected as external conversion trigger for DAC channel

DAC_TRIGGER_HR1_TRGO2

HR1 TRGO2 selected as external conversion trigger for DAC channel

DAC_TRIGGER_LP1_OUT

LP1 OUT TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_LP2_OUT

LP2 OUT TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_EXT_IT9

EXTI Line9 event selected as external conversion trigger for DAC channel

DAC User Trimming**DAC_TRIMMING_FACTORY**

Factory trimming

DAC_TRIMMING_USER

User trimming

15 HAL DAC Extension Driver

15.1 DACEx Firmware driver API description

15.1.1 How to use this driver

- When Dual mode is enabled (i.e DAC Channel1 and Channel2 are used simultaneously) : Use HAL_DACEx_DualGetValue() to get digital data to be converted and use HAL_DACEx_DualSetValue() to set digital value to converted simultaneously in Channel 1 and Channel 2.
- Use HAL_DACEx_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL_DACEx_NoiseWaveGenerate() to generate Noise signal.
- HAL_DACEx_SelfCalibrate to calibrate one DAC channel.
- HAL_DACEx_SetUserTrimming to set user trimming value.
- HAL_DACEx_GetTrimOffset to retrieve trimming value (factory setting after reset, user setting if HAL_DACEx_SetUserTrimming have been used at least one time after reset).

15.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion with triangle wave generation.
- Start conversion with noise wave generation.
- Start self calibration.
- Set user trimming mode.
- Get result of dual mode conversion.

This section contains the following APIs:

- [**HAL_DACEx_TriangleWaveGenerate**](#)
- [**HAL_DACEx_NoiseWaveGenerate**](#)
- [**HAL_DACEx_DualSetValue**](#)
- [**HAL_DACEx_ConvCpltCallbackCh2**](#)
- [**HAL_DACEx_ConvHalfCpltCallbackCh2**](#)
- [**HAL_DACEx_ErrorCallbackCh2**](#)
- [**HAL_DACEx_DMAUnderrunCallbackCh2**](#)
- [**HAL_DACEx_SelfCalibrate**](#)
- [**HAL_DACEx_SetUserTrimming**](#)

15.1.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- [**HAL_DACEx_DualGetValue**](#)
- [**HAL_DACEx_GetTrimOffset**](#)

15.1.4 Detailed description of functions

HAL_DACEx_TriangleWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:
 - DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1
 - DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3
 - DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7
 - DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15
 - DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31
 - DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63
 - DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127
 - DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255
 - DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511
 - DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023
 - DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047
 - DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095

Return values

- **HAL:** status

HAL_DACEx_NoiseWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected
- **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:
 - DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation
 - DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation
 - DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation

- DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation
- DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation
- DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation
- DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
- DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
- DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
- DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
- DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
- DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values

- **HAL:** status

HAL_DACEx_DualSetValue

Function name

```
HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment,  
uint32_t Data1, uint32_t Data2)
```

Function description

Set the specified data holding register value for dual DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data1:** Data for DAC Channel2 to be loaded in the selected data holding register.
- **Data2:** Data for DAC Channel1 to be loaded in the selected data holding register.

Return values

- **HAL:** status

Notes

- In dual mode, a unique register access is required to write in both DAC channels at the same time.

HAL_DACEx_ConvCpltCallbackCh2

Function name

```
void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)
```

Function description

Conversion complete callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ConvHalfCpltCallbackCh2

Function name

```
void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)
```

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ErrorCallbackCh2

Function name

void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_DMAUnderrunCallbackCh2

Function name

void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_DACEx_SelfCalibrate (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Run the self calibration of one DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC channel configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **Updates:** DAC_TrimmingValue. , DAC_UserTrimming set to DAC_UserTrimming
- **HAL:** status

Notes

- Calibration runs about 7 ms.

HAL_DACEx_SetUserTrimming

Function name

```
HAL_StatusTypeDef HAL_DACEx_SetUserTrimming (DAC_HandleTypeDef * hdac,  
DAC_ChannelConfTypeDef * sConfig, uint32_t Channel, uint32_t NewTrimmingValue)
```

Function description

Set the trimming mode and trimming value (user trimming mode applied).

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure updated with new DAC trimming value.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected
- **NewTrimmingValue:** DAC new trimming value

Return values

- **HAL:** status

HAL_DACEx_DualGetValue

Function name

```
uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)
```

Function description

Return the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **The:** selected DAC channel data output value.

HAL_DACEx_GetTrimOffset

Function name

```
uint32_t HAL_DACEx_GetTrimOffset (DAC_HandleTypeDef * hdac, uint32_t Channel)
```

Function description

Return the DAC trimming value.

Parameters

- **hdac:** : DAC handle
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
 - DAC_CHANNEL_1: DAC Channel1 selected
 - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **Trimming:** value : range: 0->31

DAC_DMAConvCpltCh2**Function name**

```
void DAC_DMAConvCpltCh2 (DMA_HandleTypeDef * hdma)
```

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh2**Function name**

```
void DAC_DMAErrorCh2 (DMA_HandleTypeDef * hdma)
```

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAMhalfConvCpltCh2**Function name**

```
void DAC_DMAMhalfConvCpltCh2 (DMA_HandleTypeDef * hdma)
```

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

15.2 DACEEx Firmware driver defines

15.2.1 DACEEx

DACEEx lfsrunmask triangle amplitude

DAC_LFSRUNMASK_BIT0

Unmask DAC channel LFSR bit0 for noise wave generation

DAC_LFSRUNMASK_BITS1_0

Unmask DAC channel LFSR bit[1:0] for noise wave generation

DAC_LFSRUNMASK_BITS2_0

Unmask DAC channel LFSR bit[2:0] for noise wave generation

DAC_LFSRUNMASK_BITS3_0

Unmask DAC channel LFSR bit[3:0] for noise wave generation

DAC_LFSRUNMASK_BITS4_0

Unmask DAC channel LFSR bit[4:0] for noise wave generation

DAC_LFSRUNMASK_BITS5_0

Unmask DAC channel LFSR bit[5:0] for noise wave generation

DAC_LFSRUNMASK_BITS6_0

Unmask DAC channel LFSR bit[6:0] for noise wave generation

DAC_LFSRUNMASK_BITS7_0

Unmask DAC channel LFSR bit[7:0] for noise wave generation

DAC_LFSRUNMASK_BITS8_0

Unmask DAC channel LFSR bit[8:0] for noise wave generation

DAC_LFSRUNMASK_BITS9_0

Unmask DAC channel LFSR bit[9:0] for noise wave generation

DAC_LFSRUNMASK_BITS10_0

Unmask DAC channel LFSR bit[10:0] for noise wave generation

DAC_LFSRUNMASK_BITS11_0

Unmask DAC channel LFSR bit[11:0] for noise wave generation

DAC_TRIANGLEAMPLITUDE_1

Select max triangle amplitude of 1

DAC_TRIANGLEAMPLITUDE_3

Select max triangle amplitude of 3

DAC_TRIANGLEAMPLITUDE_7

Select max triangle amplitude of 7

DAC_TRIANGLEAMPLITUDE_15

Select max triangle amplitude of 15

DAC_TRIANGLEAMPLITUDE_31

Select max triangle amplitude of 31

DAC_TRIANGLEAMPLITUDE_63

Select max triangle amplitude of 63

DAC_TRIANGLEAMPLITUDE_127

Select max triangle amplitude of 127

DAC_TRIANGLEAMPLITUDE_255

Select max triangle amplitude of 255

DAC_TRIANGLEAMPLITUDE_511

Select max triangle amplitude of 511

DAC_TRIANGLEAMPLITUDE_1023

Select max triangle amplitude of 1023

DAC_TRIANGLEAMPLITUDE_2047

Select max triangle amplitude of 2047

DAC_TRIANGLEAMPLITUDE_4095

Select max triangle amplitude of 4095

16 HAL DCMI Generic Driver

16.1 DCMI Firmware driver registers structures

16.1.1 DCMI_CodesInitTypeDef

Data Fields

- `uint8_t FrameStartCode`
- `uint8_t LineStartCode`
- `uint8_t LineEndCode`
- `uint8_t FrameEndCode`

Field Documentation

- `uint8_t DCMI_CodesInitTypeDef::FrameStartCode`
Specifies the code of the frame start delimiter.
- `uint8_t DCMI_CodesInitTypeDef::LineStartCode`
Specifies the code of the line start delimiter.
- `uint8_t DCMI_CodesInitTypeDef::LineEndCode`
Specifies the code of the line end delimiter.
- `uint8_t DCMI_CodesInitTypeDef::FrameEndCode`
Specifies the code of the frame end delimiter.

16.1.2 DCMI_InitTypeDef

Data Fields

- `uint32_t SyncroMode`
- `uint32_t PCKPolarity`
- `uint32_t VSPolarity`
- `uint32_t HSPolarity`
- `uint32_t CaptureRate`
- `uint32_t ExtendedDataMode`
- `DCMI_CodesInitTypeDef SyncroCode`
- `uint32_t JPEGMode`
- `uint32_t ByteSelectMode`
- `uint32_t ByteSelectStart`
- `uint32_t LineSelectMode`
- `uint32_t LineSelectStart`

Field Documentation

- `uint32_t DCMI_InitTypeDef::SyncroMode`
Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of `DCMI Synchronization Mode`
- `uint32_t DCMI_InitTypeDef::PCKPolarity`
Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of `DCMI PIXCK Polarity`
- `uint32_t DCMI_InitTypeDef::VSPolarity`

Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of **DCMI VSYNC Polarity**

- **`uint32_t DCMI_InitTypeDef::HSPolarity`**

Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of **DCMI HSYNC Polarity**

- **`uint32_t DCMI_InitTypeDef::CaptureRate`**

Specifies the frequency of frame capture: All, 1/2 or 1/4. This parameter can be a value of **DCMI Capture Rate**

- **`uint32_t DCMI_InitTypeDef::ExtendedDataMode`**

Specifies the data width: 8-bit, 10-bit, 12-bit or 14-bit. This parameter can be a value of **DCMI Extended Data Mode**

- **`DCMI_CodesInitTypeDef DCMI_InitTypeDef::SyncroCode`**

Specifies the code of the line/frame start delimiter and the line/frame end delimiter

- **`uint32_t DCMI_InitTypeDef::JPEGMode`**

Enable or Disable the JPEG mode. This parameter can be a value of **DCMI MODE JPEG**

- **`uint32_t DCMI_InitTypeDef::ByteSelectMode`**

Specifies the data to be captured by the interface This parameter can be a value of **DCMI Byte Select Mode**

- **`uint32_t DCMI_InitTypeDef::ByteSelectStart`**

Specifies if the data to be captured by the interface is even or odd This parameter can be a value of **DCMI Byte Select Start**

- **`uint32_t DCMI_InitTypeDef::LineSelectMode`**

Specifies the line of data to be captured by the interface This parameter can be a value of **DCMI Line Select Mode**

- **`uint32_t DCMI_InitTypeDef::LineSelectStart`**

Specifies if the line of data to be captured by the interface is even or odd This parameter can be a value of **DCMI Line Select Start**

16.1.3 DCMI_HandleTypeDef

Data Fields

- **`DCMI_TypeDef * Instance`**
- **`DCMI_InitTypeDef Init`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_DCMI_StateTypeDef State`**
- **`__IO uint32_t XferCount`**
- **`__IO uint32_t XferSize`**
- **`uint32_t XferTransferNumber`**
- **`uint32_t pBuffPtr`**
- **`DMA_HandleTypeDef * DMA_Handle`**
- **`__IO uint32_t ErrorCode`**

Field Documentation

- **`DCMI_TypeDef* DCMI_HandleTypeDef::Instance`**

DCMI Register base address

- **`DCMI_InitTypeDef DCMI_HandleTypeDef::Init`**

DCMI parameters

- **`HAL_LockTypeDef DCMI_HandleTypeDef::Lock`**

- DCMI locking object
- `__IO HAL_DCMI_StateTypeDef DCMI_HandleTypeDef::State`
DCMI state
- `__IO uint32_t DCMI_HandleTypeDef::XferCount`
DMA transfer counter
- `__IO uint32_t DCMI_HandleTypeDef::XferSize`
DMA transfer size
- `uint32_t DCMI_HandleTypeDef::XferTransferNumber`
DMA transfer number
- `uint32_t DCMI_HandleTypeDef::pBuffPtr`
Pointer to DMA output buffer
- `DMA_HandleTypeDef* DCMI_HandleTypeDef::DMA_Handle`
Pointer to the DMA handler
- `__IO uint32_t DCMI_HandleTypeDef::ErrorCode`
DCMI Error code

16.2 DCMI Firmware driver API description

16.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMI Interface. This sequence does not take into account the configuration of the camera module, which should be made before to configure and enable the DCMI to capture images.

1. Program the required configuration through following parameters: horizontal and vertical polarity, pixel clock polarity, Capture Rate, Synchronization Mode, code of the frame delimiter and data width using `HAL_DCMI_Init()` function.
2. Configure the selected DMA stream to transfer Data from DCMI DR register to the destination memory buffer.
3. Program the required configuration through following parameters: DCMI mode, destination memory Buffer address and the data length and enable capture using `HAL_DCMI_Start_DMA()` function.
4. Optionally, configure and Enable the CROP feature to select a rectangular window from the received image using `HAL_DCMI_ConfigCrop()` and `HAL_DCMI_EnableCrop()` functions
5. The capture can be stopped using `HAL_DCMI_Stop()` function.
6. To control DCMI state you can use the function `HAL_DCMI_GetState()`.

DCMI HAL driver macros list

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

- `__HAL_DCMI_ENABLE`: Enable the DCMI peripheral.
- `__HAL_DCMI_DISABLE`: Disable the DCMI peripheral.
- `__HAL_DCMI_GET_FLAG`: Get the DCMI pending flags.
- `__HAL_DCMI_CLEAR_FLAG`: Clear the DCMI pending flags.
- `__HAL_DCMI_ENABLE_IT`: Enable the specified DCMI interrupts.
- `__HAL_DCMI_DISABLE_IT`: Disable the specified DCMI interrupts.
- `__HAL_DCMI_GET_IT_SOURCE`: Check whether the specified DCMI interrupt has occurred or not.

Note: You can refer to the DCMI HAL driver header file for more useful macros

16.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI

This section contains the following APIs:

- [*HAL_DCMI_Init*](#)
- [*HAL_DCMI_DelInit*](#)
- [*HAL_DCMI_MspInit*](#)
- [*HAL_DCMI_MspDelInit*](#)

16.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length and Enables DCMI DMA request and enables DCMI capture
- Stop the DCMI capture.
- Handles DCMI interrupt request.

This section contains the following APIs:

- [*HAL_DCMI_Start_DMA*](#)
- [*HAL_DCMI_Stop*](#)
- [*HAL_DCMI_Suspend*](#)
- [*HAL_DCMI_Resume*](#)
- [*HAL_DCMI_IRQHandler*](#)
- [*HAL_DCMI_ErrorCallback*](#)
- [*HAL_DCMI_LineEventCallback*](#)
- [*HAL_DCMI_VsyncEventCallback*](#)
- [*HAL_DCMI_FrameEventCallback*](#)

16.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the CROP feature.
- Enable/Disable the CROP feature.

This section contains the following APIs:

- [*HAL_DCMI_ConfigCrop*](#)
- [*HAL_DCMI_DisableCrop*](#)
- [*HAL_DCMI_EnableCrop*](#)

16.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DCMI state.
- Get the specific DCMI error flag.

This section contains the following APIs:

- [*HAL_DCMI_GetState*](#)
- [*HAL_DCMI_GetError*](#)

16.2.6 Detailed description of functions

HAL_DCMI_Init

Function name

HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef * hdcmi)

Function description

Initializes the DCMI according to the specified parameters in the DCMI_InitTypeDef and create the associated handle.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_DeInit

Function name

HAL_StatusTypeDef HAL_DCMI_DeInit (DCMI_HandleTypeDef * hdcmi)

Function description

Deinitializes the DCMI peripheral registers to their default reset values.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_MspInit

Function name

void HAL_DCMI_MspInit (DCMI_HandleTypeDef * hdcmi)

Function description

Initializes the DCMI MSP.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_MspDeInit

Function name

void HAL_DCMI_MspDeInit (DCMI_HandleTypeDef * hdcmi)

Function description

Deinitializes the DCMI MSP.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_Start_DMA

Function name

HAL_StatusTypeDef HAL_DCMI_Start_DMA (DCMI_HandleTypeDef * hdcmi, uint32_t DCMI_Mode, uint32_t pData, uint32_t Length)

Function description

Enables DCMI DMA request and enables DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **DCMI_Mode:** DCMI capture mode snapshot or continuous grab.
- **pData:** The destination memory Buffer address (LCD Frame buffer).
- **Length:** The length of capture to be transferred.

Return values

- **HAL:** status

HAL_DCMI_Stop

Function name

HAL_StatusTypeDef HAL_DCMI_Stop (DCMI_HandleTypeDef * hdcmi)

Function description

Disable DCMI DMA request and Disable DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_Suspend

Function name

HAL_StatusTypeDef HAL_DCMI_Suspend (DCMI_HandleTypeDef * hdcmi)

Function description

Suspend DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_Resume

Function name

HAL_StatusTypeDef HAL_DCMI_Resume (DCMI_HandleTypeDef * hdcmi)

Function description

Resume DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_ErrorCallback**Function name**

```
void HAL_DCMI_ErrorCallback (DCMI_HandleTypeDef * hdcmi)
```

Function description

Error DCMI callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_LineEventCallback**Function name**

```
void HAL_DCMI_LineEventCallback (DCMI_HandleTypeDef * hdcmi)
```

Function description

Line Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_FrameEventCallback**Function name**

```
void HAL_DCMI_FrameEventCallback (DCMI_HandleTypeDef * hdcmi)
```

Function description

Frame Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_VsyncEventCallback**Function name**

```
void HAL_DCMI_VsyncEventCallback (DCMI_HandleTypeDef * hdcmi)
```

Function description

VSYNC Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_IRQHandler

Function name

```
void HAL_DCMI_IRQHandler (DCMI_HandleTypeDef * hdcmi)
```

Function description

Handles DCMI interrupt request.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for the DCMI.

Return values

- **None:**

HAL_DCMI_ConfigCrop

Function name

```
HAL_StatusTypeDef HAL_DCMI_ConfigCrop (DCMI_HandleTypeDef * hdcmi, uint32_t X0, uint32_t Y0,  
uint32_t XSize, uint32_t YSize)
```

Function description

Configure the DCMI CROP coordinate.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **YSize:** DCMI Line number
- **XSize:** DCMI Pixel per line
- **X0:** DCMI window X offset
- **Y0:** DCMI window Y offset

Return values

- **HAL:** status

HAL_DCMI_EnableCrop

Function name

```
HAL_StatusTypeDef HAL_DCMI_EnableCrop (DCMI_HandleTypeDef * hdcmi)
```

Function description

Enable the Crop feature.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_DisableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_DisableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

Disable the Crop feature.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_GetState

Function name

HAL_DCMI_StateTypeDef HAL_DCMI_GetState (DCMI_HandleTypeDef * hdcmi)

Function description

Return the DCMI state.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** state

HAL_DCMI_GetError

Function name

uint32_t HAL_DCMI_GetError (DCMI_HandleTypeDef * hdcmi)

Function description

Return the DCMI error code.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **DCMI:** Error Code

16.3 DCMI Firmware driver defines

16.3.1 DCMI

DCMI Byte Select Mode

DCMI_BSM_ALL

Interface captures all received data

DCMI_BSM_OTHER

Interface captures every other byte from the received data

DCMI_BSM_ALTERNATE_4

Interface captures one byte out of four

DCMI_BSM_ALTERNATE_2

Interface captures two bytes out of four

DCMI Byte Select Start**DCMI_OEBS_ODD**

Interface captures first data from the frame/line start, second one being dropped

DCMI_OEBS_EVEN

Interface captures second data from the frame/line start, first one being dropped

DCMI Capture Mode**DCMI_MODE_CONTINUOUS**

The received data are transferred continuously into the destination memory through the DMA

DCMI_MODE_SNAPSHOT

Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA

DCMI Capture Rate**DCMI_CR_ALL_FRAME**

All frames are captured

DCMI_CR_ALTERNATE_2_FRAME

Every alternate frame captured

DCMI_CR_ALTERNATE_4_FRAME

One frame in 4 frames captured

DCMI Error Code**HAL_DCMI_ERROR_NONE**

No error

HAL_DCMI_ERROR_OVR

Overrun error

HAL_DCMI_ERROR_SYNC

Synchronization error

HAL_DCMI_ERROR_TIMEOUT

Timeout error

HAL_DCMI_ERROR_DMA

DMA error

DCMI Exported Macros**_HAL_DCMI_RESET_HANDLE_STATE****Description:**

- Reset DCMI handle state.

Parameters:

- _HANDLE__: specifies the DCMI handle.

Return value:

- None

[__HAL_DCMI_ENABLE](#)**Description:**

- Enable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

[__HAL_DCMI_DISABLE](#)**Description:**

- Disable the DCMI.

Parameters:

- __HANDLE__: DCMI handle

Return value:

- None

[__HAL_DCMI_GET_FLAG](#)**Description:**

- Get the DCMI pending flag.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: Get the specified flag. This parameter can be one of the following values (no combination allowed)
 - DCMI_FLAG_HSYNC: HSYNC pin state (active line / synchronization between lines)
 - DCMI_FLAG_VSYNC: VSYNC pin state (active frame / synchronization between frames)
 - DCMI_FLAG_FNE: FIFO empty flag
 - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
 - DCMI_FLAG_OVRRI: Overrun flag mask
 - DCMI_FLAG_ERRRI: Synchronization error flag mask
 - DCMI_FLAG_VSYNCRI: VSYNC flag mask
 - DCMI_FLAG_LINERI: Line flag mask
 - DCMI_FLAG_FRAMEMI: DCMI Capture complete masked interrupt status
 - DCMI_FLAG_OVRMI: DCMI Overrun masked interrupt status
 - DCMI_FLAG_ERRMI: DCMI Synchronization error masked interrupt status
 - DCMI_FLAG_VSYNCFMI: DCMI VSYNC masked interrupt status
 - DCMI_FLAG_LINEMI: DCMI Line masked interrupt status

Return value:

- The: state of FLAG.

[__HAL_DCMI_CLEAR_FLAG](#)**Description:**

- Clear the DCMI pending flags.

Parameters:

- __HANDLE__: DCMI handle

- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
 - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
 - DCMI_FLAG_OVFRI: Overflow flag mask
 - DCMI_FLAG_ERRRI: Synchronization error flag mask
 - DCMI_FLAG_VSYNCRI: VSYNC flag mask
 - DCMI_FLAG_LINERI: Line flag mask

Return value:

- None

[__HAL_DCMI_ENABLE_IT](#)**Description:**

- Enable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
 - DCMI_IT_FRAME: Frame capture complete interrupt mask
 - DCMI_IT_OVF: Overflow interrupt mask
 - DCMI_IT_ERR: Synchronization error interrupt mask
 - DCMI_IT_VSYNC: VSYNC interrupt mask
 - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

[__HAL_DCMI_DISABLE_IT](#)**Description:**

- Disable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
 - DCMI_IT_FRAME: Frame capture complete interrupt mask
 - DCMI_IT_OVF: Overflow interrupt mask
 - DCMI_IT_ERR: Synchronization error interrupt mask
 - DCMI_IT_VSYNC: VSYNC interrupt mask
 - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

[__HAL_DCMI_GET_IT_SOURCE](#)**Description:**

- Check whether the specified DCMI interrupt has occurred or not.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt source to check. This parameter can be one of the following values:
 - DCMI_IT_FRAME: Frame capture complete interrupt mask
 - DCMI_IT_OVF: Overflow interrupt mask

- DCMI_IT_ERR: Synchronization error interrupt mask
- DCMI_IT_VSYNC: VSYNC interrupt mask
- DCMI_IT_LINE: Line interrupt mask

Return value:

- The state of INTERRUPT.

DCMI Extended Data Mode**DCMI_EXTEND_DATA_8B**

Interface captures 8-bit data on every pixel clock

DCMI_EXTEND_DATA_10B

Interface captures 10-bit data on every pixel clock

DCMI_EXTEND_DATA_12B

Interface captures 12-bit data on every pixel clock

DCMI_EXTEND_DATA_14B

Interface captures 14-bit data on every pixel clock

DCMI Flags**DCMI_FLAG_HSYNC**

Hsync pin state (active line / synchronization between lines)

DCMI_FLAG_VSYNC

Vsync pin state (active frame / synchronization between frames)

DCMI_FLAG_FNE

FIFO not empty flag

DCMI_FLAG_FRAMERI

Frame capture complete interrupt flag

DCMI_FLAG_OVRRI

Overrun interrupt flag

DCMI_FLAG_ERRRI

Synchronization error interrupt flag

DCMI_FLAG_VSYNCRI

Vsync interrupt flag

DCMI_FLAG_LINERI

Line interrupt flag

DCMI_FLAG_FRAMEMI

DCMI Frame capture complete masked interrupt status

DCMI_FLAG_OVRMI

DCMI Overrun masked interrupt status

DCMI_FLAG_ERRMI

DCMI Synchronization error masked interrupt status

DCMI_FLAG_VSYNCMI

DCMI VSYNC masked interrupt status

DCMI_FLAG_LINEMI

DCMI Line masked interrupt status

DCMI HSYNC Polarity**DCMI_HSPOLARITY_LOW**

Horizontal synchronization active Low

DCMI_HSPOLARITY_HIGH

Horizontal synchronization active High

DCMI interrupt sources**DCMI_IT_FRAME**

Capture complete interrupt

DCMI_IT_OVR

Overrun interrupt

DCMI_IT_ERR

Synchronization error interrupt

DCMI_IT_VSYNC

VSYNC interrupt

DCMI_IT_LINE

Line interrupt

DCMI Line Select Mode**DCMI_LSM_ALL**

Interface captures all received lines

DCMI_LSM_ALTERNATE_2

Interface captures one line out of two

DCMI Line Select Start**DCMI_OELS_ODD**

Interface captures first line from the frame start, second one being dropped

DCMI_OELS_EVEN

Interface captures second line from the frame start, first one being dropped

DCMI MODE JPEG**DCMI_JPEG_DISABLE**

Mode JPEG Disabled

DCMI_JPEG_ENABLE

Mode JPEG Enabled

DCMI PIXCK Polarity

DCMI_PCKPOLARITY_FALLING

Pixel clock active on Falling edge

DCMI_PCKPOLARITY_RISING

Pixel clock active on Rising edge

DCMI Synchronization Mode**DCMI_SYNCHRO_HARDWARE**

Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals

DCMI_SYNCHRO_EMBEDDED

Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

DCMI VSYNC Polarity**DCMI_VSPOLARITY_LOW**

Vertical synchronization active Low

DCMI_VSPOLARITY_HIGH

Vertical synchronization active High

DCMI Window Coordinate**DCMI_WINDOW_COORDINATE**

Window coordinate

DCMI Window Height**DCMI_WINDOW_HEIGHT**

Window Height

17 HAL DFSDM Generic Driver

17.1 DFSDM Firmware driver registers structures

17.1.1 DFSDM_Channel_OutputClockTypeDef

Data Fields

- *FunctionalState Activation*
- *uint32_t Selection*
- *uint32_t Divider*

Field Documentation

- *FunctionalState DFSDM_Channel_OutputClockTypeDef::Activation*
Output clock enable/disable
- *uint32_t DFSDM_Channel_OutputClockTypeDef::Selection*
Output clock is system clock or audio clock. This parameter can be a value of **DFSDM channel output clock selection**
- *uint32_t DFSDM_Channel_OutputClockTypeDef::Divider*
Output clock divider. This parameter must be a number between Min_Data = 2 and Max_Data = 256

17.1.2 DFSDM_Channel_InputTypeDef

Data Fields

- *uint32_t Multiplexer*
- *uint32_t DataPacking*
- *uint32_t Pins*

Field Documentation

- *uint32_t DFSDM_Channel_InputTypeDef::Multiplexer*
Input is external serial inputs ,internal register or ADC output. This parameter can be a value of **DFSDM channel input multiplexer**
- *uint32_t DFSDM_Channel_InputTypeDef::DataPacking*
Standard, interleaved or dual mode for internal register. This parameter can be a value of **DFSDM channel input data packing**
- *uint32_t DFSDM_Channel_InputTypeDef::Pins*
Input pins are taken from same or following channel. This parameter can be a value of **DFSDM channel input pins**

17.1.3 DFSDM_Channel_SerialInterfaceTypeDef

Data Fields

- *uint32_t Type*
- *uint32_t SpiClock*

Field Documentation

- *uint32_t DFSDM_Channel_SerialInterfaceTypeDef::Type*

SPI or Manchester modes. This parameter can be a value of **DFSDM channel serial interface type**

- **`uint32_t DFSDM_Channel_SerialInterfaceTypeDef::SpiClock`**

SPI clock select (external or internal with different sampling point). This parameter can be a value of **DFSDM channel SPI clock selection**

17.1.4 DFSDM_Channel_AwdTypeDef

Data Fields

- **`uint32_t FilterOrder`**
- **`uint32_t Oversampling`**

Field Documentation

- **`uint32_t DFSDM_Channel_AwdTypeDef::FilterOrder`**

Analog watchdog Sinc filter order. This parameter can be a value of **DFSDM channel analog watchdog filter order**

- **`uint32_t DFSDM_Channel_AwdTypeDef::Oversampling`**

Analog watchdog filter oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 32

17.1.5 DFSDM_Channel_InitTypeDef

Data Fields

- **`DFSDM_Channel_OutputClockTypeDef OutputClock`**
- **`DFSDM_Channel_InputTypeDef Input`**
- **`DFSDM_Channel_SerialInterfaceTypeDef SerialInterface`**
- **`DFSDM_Channel_AwdTypeDef Awd`**
- **`int32_t Offset`**
- **`uint32_t RightBitShift`**

Field Documentation

- **`DFSDM_Channel_OutputClockTypeDef DFSDM_Channel_InitTypeDef::OutputClock`**

DFSDM channel output clock parameters

- **`DFSDM_Channel_InputTypeDef DFSDM_Channel_InitTypeDef::Input`**

DFSDM channel input parameters

- **`DFSDM_Channel_SerialInterfaceTypeDef DFSDM_Channel_InitTypeDef::SerialInterface`**

DFSDM channel serial interface parameters

- **`DFSDM_Channel_AwdTypeDef DFSDM_Channel_InitTypeDef::Awd`**

DFSDM channel analog watchdog parameters

- **`int32_t DFSDM_Channel_InitTypeDef::Offset`**

DFSDM channel offset. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607

- **`uint32_t DFSDM_Channel_InitTypeDef::RightBitShift`**

DFSDM channel right bit shift. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F

17.1.6 DFSDM_Channel_HandleTypeDef

Data Fields

- **`DFSDM_Channel_TypeDef * Instance`**

- *DFSDM_Channel_InitTypeDef Init*
- *HAL_DFSDM_Channel_StateTypeDef State*

Field Documentation

- *DFSDM_Channel_TypeDef* DFSDM_Channel_HandleTypeDef::Instance*
DFSDM channel instance
- *DFSDM_Channel_InitTypeDef DFSDM_Channel_HandleTypeDef::Init*
DFSDM channel init parameters
- *HAL_DFSDM_Channel_StateTypeDef DFSDM_Channel_HandleTypeDef::State*
DFSDM channel state

17.1.7 DFSDM_Filter-RegularParamTypeDef

Data Fields

- *uint32_t Trigger*
- *FunctionalState FastMode*
- *FunctionalState DmaMode*

Field Documentation

- *uint32_t DFSDM_Filter-RegularParamTypeDef::Trigger*
Trigger used to start regular conversion: software or synchronous. This parameter can be a value of **DFSDM filter conversion trigger**
- *FunctionalState DFSDM_Filter-RegularParamTypeDef::FastMode*
Enable/disable fast mode for regular conversion
- *FunctionalState DFSDM_Filter-RegularParamTypeDef::DmaMode*
Enable/disable DMA for regular conversion

17.1.8 DFSDM_Filter-InjectedParamTypeDef

Data Fields

- *uint32_t Trigger*
- *FunctionalState ScanMode*
- *FunctionalState DmaMode*
- *uint32_t ExtTrigger*
- *uint32_t ExtTriggerEdge*

Field Documentation

- *uint32_t DFSDM_Filter-InjectedParamTypeDef::Trigger*
Trigger used to start injected conversion: software, external or synchronous. This parameter can be a value of **DFSDM filter conversion trigger**
- *FunctionalState DFSDM_Filter-InjectedParamTypeDef::ScanMode*
Enable/disable scanning mode for injected conversion
- *FunctionalState DFSDM_Filter-InjectedParamTypeDef::DmaMode*
Enable/disable DMA for injected conversion
- *uint32_t DFSDM_Filter_InjectedParamTypeDef::ExtTrigger*
External trigger. This parameter can be a value of **DFSDM filter external trigger**
- *uint32_t DFSDM_Filter_InjectedParamTypeDef::ExtTriggerEdge*

External trigger edge: rising, falling or both. This parameter can be a value of [DFSDM filter external trigger edge](#)

17.1.9 DFSDM_Filter_FilterParamTypeDef

Data Fields

- `uint32_t SincOrder`
- `uint32_t Oversampling`
- `uint32_t IntOversampling`

Field Documentation

- `uint32_t DFSDM_Filter_FilterParamTypeDef::SincOrder`
Sinc filter order. This parameter can be a value of [DFSDM filter sinc order](#)
- `uint32_t DFSDM_Filter_FilterParamTypeDef::Oversampling`
Filter oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 1024
- `uint32_t DFSDM_Filter_FilterParamTypeDef::IntOversampling`
Integrator oversampling ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 256

17.1.10 DFSDM_Filter_InitTypeDef

Data Fields

- `DFSDM_Filter-RegularParamTypeDef RegularParam`
- `DFSDM_Filter-InjectedParamTypeDef InjectedParam`
- `DFSDM_Filter-FilterParamTypeDef FilterParam`

Field Documentation

- `DFSDM_Filter-RegularParamTypeDef DFSDM_Filter_InitTypeDef::RegularParam`
DFSDM regular conversion parameters
- `DFSDM_Filter-InjectedParamTypeDef DFSDM_Filter_InitTypeDef::InjectedParam`
DFSDM injected conversion parameters
- `DFSDM_Filter-FilterParamTypeDef DFSDM_Filter_InitTypeDef::FilterParam`
DFSDM filter parameters

17.1.11 DFSDM_Filter_HandleTypeDef

Data Fields

- `DFSDM_Filter_TypeDef * Instance`
- `DFSDM_Filter_InitTypeDef Init`
- `DMA_HandleTypeDef * hdmaReg`
- `DMA_HandleTypeDef * hdmaInj`
- `uint32_t RegularContMode`
- `uint32_t RegularTrigger`
- `uint32_t InjectedTrigger`
- `uint32_t ExtTriggerEdge`
- `FunctionalState InjectedScanMode`
- `uint32_t InjectedChannelsNbr`
- `uint32_t InjConvRemaining`
- `HAL_DFSDM_Filter_StateTypeDef State`

- `uint32_t ErrorCode`

Field Documentation

- `DFSDM_Filter_TypeDef* DFSDM_Filter_HandleTypeDef::Instance`
DFSDM filter instance
- `DFSDM_Filter_InitTypeDef DFSDM_Filter_HandleTypeDef::Init`
DFSDM filter init parameters
- `DMA_HandleTypeDef* DFSDM_Filter_HandleTypeDef::hdmaReg`
Pointer on DMA handler for regular conversions
- `DMA_HandleTypeDef* DFSDM_Filter_HandleTypeDef::hdmaInj`
Pointer on DMA handler for injected conversions
- `uint32_t DFSDM_Filter_HandleTypeDef::RegularContMode`
Regular conversion continuous mode
- `uint32_t DFSDM_Filter_HandleTypeDef::RegularTrigger`
Trigger used for regular conversion
- `uint32_t DFSDM_Filter_HandleTypeDef::InjectedTrigger`
Trigger used for injected conversion
- `uint32_t DFSDM_Filter_HandleTypeDef::ExtTriggerEdge`
Rising, falling or both edges selected
- `FunctionalState DFSDM_Filter_HandleTypeDef::InjectedScanMode`
Injected scanning mode
- `uint32_t DFSDM_Filter_HandleTypeDef::InjectedChannelsNbr`
Number of channels in injected sequence
- `uint32_t DFSDM_Filter_HandleTypeDef::InjConvRemaining`
Injected conversions remaining
- `HAL_DFSDM_Filter_StateTypeDef DFSDM_Filter_HandleTypeDef::State`
DFSDM filter state
- `uint32_t DFSDM_Filter_HandleTypeDef::ErrorCode`
DFSDM filter error code

17.1.12 DFSDM_Filter_AwdParamTypeDef

Data Fields

- `uint32_t DataSource`
- `uint32_t Channel`
- `int32_t HighThreshold`
- `int32_t LowThreshold`
- `uint32_t HighBreakSignal`
- `uint32_t LowBreakSignal`

Field Documentation

- `uint32_t DFSDM_Filter_AwdParamTypeDef::DataSource`
Values from digital filter or from channel watchdog filter. This parameter can be a value of `DFSDM filter analog watchdog data source`
- `uint32_t DFSDM_Filter_AwdParamTypeDef::Channel`

Analog watchdog channel selection. This parameter can be a values combination of **DFSDM Channel Selection**

- ***int32_t DFSDM_Filter_AwdParamTypeDef::HighThreshold***
High threshold for the analog watchdog. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607
- ***int32_t DFSDM_Filter_AwdParamTypeDef::LowThreshold***
Low threshold for the analog watchdog. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607
- ***uint32_t DFSDM_Filter_AwdParamTypeDef::HighBreakSignal***
Break signal assigned to analog watchdog high threshold event. This parameter can be a values combination of **DFSDM break signals**
- ***uint32_t DFSDM_Filter_AwdParamTypeDef::LowBreakSignal***
Break signal assigned to analog watchdog low threshold event. This parameter can be a values combination of **DFSDM break signals**

17.2 DFSDM Firmware driver API description

17.2.1 How to use this driver

Channel initialization

1. User has first to initialize channels (before filters initialization).
2. As prerequisite, fill in the HAL_DFSDM_ChannelMspInit() :
 - Enable DFSDMz clock interface with __HAL_RCC_DFSDMz_CLK_ENABLE().
 - Enable the clocks for the DFSDMz GPIOs with __HAL_RCC_GPIOx_CLK_ENABLE().
 - Configure these DFSDMz pins in alternate mode using HAL_GPIO_Init().
 - If interrupt mode is used, enable and configure DFSDMz_FLT0 global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
3. Configure the output clock, input, serial interface, analog watchdog, offset and data right bit shift parameters for this channel using the HAL_DFSDM_ChannellInit() function.

Channel clock absence detector

1. Start clock absence detector using HAL_DFSDM_ChannelCkabStart() or HAL_DFSDM_ChannelCkabStart_IT().
2. In polling mode, use HAL_DFSDM_ChannelPollForCkab() to detect the clock absence.
3. In interrupt mode, HAL_DFSDM_ChannelCkabCallback() will be called if clock absence is detected.
4. Stop clock absence detector using HAL_DFSDM_ChannelCkabStop() or HAL_DFSDM_ChannelCkabStop_IT().
5. Please note that the same mode (polling or interrupt) has to be used for all channels because the channels are sharing the same interrupt.
6. Please note also that in interrupt mode, if clock absence detector is stopped for one channel, interrupt will be disabled for all channels.

Channel short circuit detector

1. Start short circuit detector using HAL_DFSDM_ChannelScdStart() or or HAL_DFSDM_ChannelScdStart_IT().
2. In polling mode, use HAL_DFSDM_ChannelPollForScd() to detect short circuit.
3. In interrupt mode, HAL_DFSDM_ChannelScdCallback() will be called if short circuit is detected.
4. Stop short circuit detector using HAL_DFSDM_ChannelScdStop() or or HAL_DFSDM_ChannelScdStop_IT().

5. Please note that the same mode (polling or interrupt) has to be used for all channels because the channels are sharing the same interrupt.
6. Please note also that in interrupt mode, if short circuit detector is stopped for one channel, interrupt will be disabled for all channels.

Channel analog watchdog value

1. Get analog watchdog filter value of a channel using HAL_DFSDM_ChannelGetAwdValue().

Channel offset value

1. Modify offset value of a channel using HAL_DFSDM_ChannelModifyOffset().

Filter initialization

1. After channel initialization, user has to init filters.
2. As prerequisite, fill in the HAL_DFSDM_FilterMspInit() :
 - If interrupt mode is used , enable and configure DFSDMz_FLTx global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ(). Please note that DFSDMz_FLT0 global interrupt could be already enabled if interrupt is used for channel.
 - If DMA mode is used, configure DMA with HAL_DMA_Init() and link it with DFSDMz filter handle using __HAL_LINKDMA().
3. Configure the regular conversion, injected conversion and filter parameters for this filter using the HAL_DFSDM_FilterInit() function.

Filter regular channel conversion

1. Select regular channel and enable/disable continuous mode using HAL_DFSDM_FilterConfigRegChannel().
2. Start regular conversion using HAL_DFSDM_FilterRegularStart(), HAL_DFSDM_FilterRegularStart_IT(), HAL_DFSDM_FilterRegularStart_DMA() or HAL_DFSDM_FilterRegularMsbStart_DMA().
3. In polling mode, use HAL_DFSDM_FilterPollForRegConversion() to detect the end of regular conversion.
4. In interrupt mode, HAL_DFSDM_FilterRegConvCpltCallback() will be called at the end of regular conversion.
5. Get value of regular conversion and corresponding channel using HAL_DFSDM_FilterGetRegularValue().
6. In DMA mode, HAL_DFSDM_FilterRegConvHalfCpltCallback() and HAL_DFSDM_FilterRegConvCpltCallback() will be called respectively at the half transfer and at the transfer complete. Please note that HAL_DFSDM_FilterRegConvHalfCpltCallback() will be called only in DMA circular mode.
7. Stop regular conversion using HAL_DFSDM_FilterRegularStop(), HAL_DFSDM_FilterRegularStop_IT() or HAL_DFSDM_FilterRegularStop_DMA().

Filter injected channels conversion

1. Select injected channels using HAL_DFSDM_FilterConfigInjChannel().
2. Start injected conversion using HAL_DFSDM_FilterInjectedStart(), HAL_DFSDM_FilterInjectedStart_IT(), HAL_DFSDM_FilterInjectedStart_DMA() or HAL_DFSDM_FilterInjectedMsbStart_DMA().
3. In polling mode, use HAL_DFSDM_FilterPollForInjConversion() to detect the end of injected conversion.
4. In interrupt mode, HAL_DFSDM_FilterInjConvCpltCallback() will be called at the end of injected conversion.
5. Get value of injected conversion and corresponding channel using HAL_DFSDM_FilterGetInjectedValue().
6. In DMA mode, HAL_DFSDM_FilterInjConvHalfCpltCallback() and HAL_DFSDM_FilterInjConvCpltCallback() will be called respectively at the half transfer and at the transfer complete. Please note that HAL_DFSDM_FilterInjConvCpltCallback() will be called only in DMA circular mode.
7. Stop injected conversion using HAL_DFSDM_FilterInjectedStop(), HAL_DFSDM_FilterInjectedStop_IT() or HAL_DFSDM_FilterInjectedStop_DMA().

Filter analog watchdog

1. Start filter analog watchdog using HAL_DFSDM_FilterAwdStart_IT().
2. HAL_DFSDM_FilterAwdCallback() will be called if analog watchdog occurs.

3. Stop filter analog watchdog using HAL_DFSDM_FilterAwdStop_IT().

Filter extreme detector

1. Start filter extreme detector using HAL_DFSDM_FilterExdStart().
2. Get extreme detector maximum value using HAL_DFSDM_FilterGetExdMaxValue().
3. Get extreme detector minimum value using HAL_DFSDM_FilterGetExdMinValue().
4. Start filter extreme detector using HAL_DFSDM_FilterExdStop().

Filter conversion time

1. Get conversion time value using HAL_DFSDM_FilterGetConvTimeValue().

17.2.2 Channel initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the DFSDM channel.
- De-initialize the DFSDM channel.

This section contains the following APIs:

- [*HAL_DFSDM_ChannelInit*](#)
- [*HAL_DFSDM_ChannelDelInit*](#)
- [*HAL_DFSDM_ChannelMspInit*](#)
- [*HAL_DFSDM_ChannelMspDelInit*](#)

17.2.3 Channel operation functions

This section provides functions allowing to:

- Manage clock absence detector feature.
- Manage short circuit detector feature.
- Get analog watchdog value.
- Modify offset value.

This section contains the following APIs:

- [*HAL_DFSDM_ChannelCkabStart*](#)
- [*HAL_DFSDM_ChannelPollForCkab*](#)
- [*HAL_DFSDM_ChannelCkabStop*](#)
- [*HAL_DFSDM_ChannelCkabStart_IT*](#)
- [*HAL_DFSDM_ChannelCkabStop_IT*](#)
- [*HAL_DFSDM_ChannelScdStart*](#)
- [*HAL_DFSDM_ChannelPollForScd*](#)
- [*HAL_DFSDM_ChannelScdStop*](#)
- [*HAL_DFSDM_ChannelScdStart_IT*](#)
- [*HAL_DFSDM_ChannelScdStop_IT*](#)
- [*HAL_DFSDM_ChannelGetAwdValue*](#)
- [*HAL_DFSDM_ChannelModifyOffset*](#)

17.2.4 Channel state function

This section provides function allowing to:

- Get channel handle state.

This section contains the following APIs:

- [*HAL_DFSDM_ChannelGetState*](#)

17.2.5 Filter initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the DFSDM filter.
- De-initialize the DFSDM filter.

This section contains the following APIs:

- [*HAL_DFSDM_FilterInit*](#)
- [*HAL_DFSDM_FilterDelInit*](#)
- [*HAL_DFSDM_FilterMspInit*](#)
- [*HAL_DFSDM_FilterMspDelInit*](#)

17.2.6 Filter control functions

This section provides functions allowing to:

- Select channel and enable/disable continuous mode for regular conversion.
- Select channels for injected conversion.

This section contains the following APIs:

- [*HAL_DFSDM_FilterConfigRegChannel*](#)
- [*HAL_DFSDM_FilterConfigInjChannel*](#)

17.2.7 Filter operation functions

This section provides functions allowing to:

- Start conversion of regular/injected channel.
- Poll for the end of regular/injected conversion.
- Stop conversion of regular/injected channel.
- Start conversion of regular/injected channel and enable interrupt.
- Call the callback functions at the end of regular/injected conversions.
- Stop conversion of regular/injected channel and disable interrupt.
- Start conversion of regular/injected channel and enable DMA transfer.
- Stop conversion of regular/injected channel and disable DMA transfer.
- Start analog watchdog and enable interrupt.
- Call the callback function when analog watchdog occurs.
- Stop analog watchdog and disable interrupt.
- Start extreme detector.
- Stop extreme detector.
- Get result of regular channel conversion.
- Get result of injected channel conversion.
- Get extreme detector maximum and minimum values.
- Get conversion time.
- Handle DFSDM interrupt request.

This section contains the following APIs:

- [*HAL_DFSDM_FilterRegularStart*](#)
- [*HAL_DFSDM_FilterPollForRegConversion*](#)
- [*HAL_DFSDM_FilterRegularStop*](#)

- [`HAL_DFSDM_FilterRegularStart_IT`](#)
- [`HAL_DFSDM_FilterRegularStop_IT`](#)
- [`HAL_DFSDM_FilterRegularStart_DMA`](#)
- [`HAL_DFSDM_FilterRegularMsbStart_DMA`](#)
- [`HAL_DFSDM_FilterRegularStop_DMA`](#)
- [`HAL_DFSDM_FilterGetRegularValue`](#)
- [`HAL_DFSDM_FilterInjectedStart`](#)
- [`HAL_DFSDM_FilterPollForInjConversion`](#)
- [`HAL_DFSDM_FilterInjectedStop`](#)
- [`HAL_DFSDM_FilterInjectedStart_IT`](#)
- [`HAL_DFSDM_FilterInjectedStop_IT`](#)
- [`HAL_DFSDM_FilterInjectedStart_DMA`](#)
- [`HAL_DFSDM_FilterInjectedMsbStart_DMA`](#)
- [`HAL_DFSDM_FilterInjectedStop_DMA`](#)
- [`HAL_DFSDM_FilterGetInjectedValue`](#)
- [`HAL_DFSDM_FilterAwdStart_IT`](#)
- [`HAL_DFSDM_FilterAwdStop_IT`](#)
- [`HAL_DFSDM_FilterExdStart`](#)
- [`HAL_DFSDM_FilterExdStop`](#)
- [`HAL_DFSDM_FilterGetExdMaxValue`](#)
- [`HAL_DFSDM_FilterGetExdMinValue`](#)
- [`HAL_DFSDM_FilterGetConvTimeValue`](#)
- [`HAL_DFSDM_IRQHandler`](#)
- [`HAL_DFSDM_FilterRegConvCpltCallback`](#)
- [`HAL_DFSDM_FilterRegConvHalfCpltCallback`](#)
- [`HAL_DFSDM_FilterInjConvCpltCallback`](#)
- [`HAL_DFSDM_FilterInjConvHalfCpltCallback`](#)
- [`HAL_DFSDM_FilterAwdCallback`](#)
- [`HAL_DFSDM_FilterErrorCallback`](#)

17.2.8 Filter state functions

This section provides functions allowing to:

- Get the DFSDM filter state.
- Get the DFSDM filter error.

This section contains the following APIs:

- [`HAL_DFSDM_FilterGetState`](#)
- [`HAL_DFSDM_FilterGetError`](#)

17.2.9 Detailed description of functions

`HAL_DFSDM_ChannellInit`

Function name

`HAL_StatusTypeDef HAL_DFSDM_ChannellInit (DFSDM_Channel_HandleTypeDef * hdfsdm_channel)`

Function description

Initialize the DFSDM channel according to the specified parameters in the `DFSDM_ChannelInitTypeDef` structure and initialize the associated handle.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **HAL:** status.

HAL_DFSDM_ChannelDelInit

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelDelInit (DFSDM_Channel_HandleTypeDef * hdfsm_channel)

Function description

De-initialize the DFSDM channel.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **HAL:** status.

HAL_DFSDM_ChannelMspInit

Function name

void HAL_DFSDM_ChannelMspInit (DFSDM_Channel_HandleTypeDef * hdfsm_channel)

Function description

Initialize the DFSDM channel MSP.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **None:**

HAL_DFSDM_ChannelMspDelInit

Function name

void HAL_DFSDM_ChannelMspDelInit (DFSDM_Channel_HandleTypeDef * hdfsm_channel)

Function description

De-initialize the DFSDM channel MSP.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **None:**

HAL_DFSDM_ChannelCkabStart

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStart (DFSDM_Channel_HandleTypeDef * hdfsm_channel)

Function description

This function allows to start clock absence detection in polling mode.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **HAL:** status

Notes

- Same mode has to be used for all channels.
- If clock is not available on this channel during 5 seconds, clock absence detection will not be activated and function will return HAL_TIMEOUT error.

HAL_DFSDM_ChannelCkabStart_IT

Function name

```
HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStart_IT (DFSDM_Channel_HandleTypeDef *  
hdfsm_channel)
```

Function description

This function allows to start clock absence detection in interrupt mode.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **HAL:** status

Notes

- Same mode has to be used for all channels.
- If clock is not available on this channel during 5 seconds, clock absence detection will not be activated and function will return HAL_TIMEOUT error.

HAL_DFSDM_ChannelCkabStop

Function name

```
HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStop (DFSDM_Channel_HandleTypeDef *  
hdfsm_channel)
```

Function description

This function allows to stop clock absence detection in polling mode.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **HAL:** status

HAL_DFSDM_ChannelCkabStop_IT

Function name

```
HAL_StatusTypeDef HAL_DFSDM_ChannelCkabStop_IT (DFSDM_Channel_HandleTypeDef *  
hdfsm_channel)
```

Function description

This function allows to stop clock absence detection in interrupt mode.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **HAL:** status

Notes

- Interrupt will be disabled for all channels

HAL_DFSDM_ChannelScdStart

Function name

```
HAL_StatusTypeDef HAL_DFSDM_ChannelScdStart (DFSDM_Channel_HandleTypeDef *  
hdfsm_channel, uint32_t Threshold, uint32_t BreakSignal)
```

Function description

This function allows to start short circuit detection in polling mode.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.
- **Threshold:** : Short circuit detector threshold. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- **BreakSignal:** : Break signals assigned to short circuit event. This parameter can be a values combination of DFSDM break signals.

Return values

- **HAL:** status

Notes

- Same mode has to be used for all channels

HAL_DFSDM_ChannelScdStart_IT

Function name

```
HAL_StatusTypeDef HAL_DFSDM_ChannelScdStart_IT (DFSDM_Channel_HandleTypeDef *  
hdfsm_channel, uint32_t Threshold, uint32_t BreakSignal)
```

Function description

This function allows to start short circuit detection in interrupt mode.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.
- **Threshold:** : Short circuit detector threshold. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- **BreakSignal:** : Break signals assigned to short circuit event. This parameter can be a values combination of DFSDM break signals.

Return values

- **HAL:** status

Notes

- Same mode has to be used for all channels

HAL_DFSDM_ChannelScdStop

Function name

```
HAL_StatusTypeDef HAL_DFSDM_ChannelScdStop (DFSDM_Channel_HandleTypeDef *  
hdfsm_channel)
```

Function description

This function allows to stop short circuit detection in polling mode.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **HAL:** status

HAL_DFSDM_ChannelScdStop_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelScdStop_IT (DFSDM_Channel_HandleTypeDef * hdfsm_channel)

Function description

This function allows to stop short circuit detection in interrupt mode.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **HAL:** status

Notes

- Interrupt will be disabled for all channels

HAL_DFSDM_ChannelGetAwdValue

Function name

int16_t HAL_DFSDM_ChannelGetAwdValue (DFSDM_Channel_HandleTypeDef * hdfsm_channel)

Function description

This function allows to get channel analog watchdog value.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **Channel:** analog watchdog value.

HAL_DFSDM_ChannelModifyOffset

Function name

HAL_StatusTypeDef HAL_DFSDM_ChannelModifyOffset (DFSDM_Channel_HandleTypeDef * hdfsm_channel, int32_t Offset)

Function description

This function allows to modify channel offset value.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.
- **Offset:** : DFSDM channel offset. This parameter must be a number between Min_Data = -8388608 and Max_Data = 8388607.

Return values

- **HAL:** status.

HAL_DFSDM_ChannelPollForCkab

Function name

```
HAL_StatusTypeDef HAL_DFSDM_ChannelPollForCkab (DFSDM_Channel_HandleTypeDef *  
hdfsm_channel, uint32_t Timeout)
```

Function description

This function allows to poll for the clock absence detection.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.
- **Timeout:** : Timeout value in milliseconds.

Return values

- **HAL:** status

HAL_DFSDM_ChannelPollForScd

Function name

```
HAL_StatusTypeDef HAL_DFSDM_ChannelPollForScd (DFSDM_Channel_HandleTypeDef *  
hdfsm_channel, uint32_t Timeout)
```

Function description

This function allows to poll for the short circuit detection.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.
- **Timeout:** : Timeout value in milliseconds.

Return values

- **HAL:** status

HAL_DFSDM_ChannelCkabCallback

Function name

```
void HAL_DFSDM_ChannelCkabCallback (DFSDM_Channel_HandleTypeDef * hdfsm_channel)
```

Function description

Clock absence detection callback.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **None:**

HAL_DFSDM_ChannelScdCallback

Function name

```
void HAL_DFSDM_ChannelScdCallback (DFSDM_Channel_HandleTypeDef * hdfsm_channel)
```

Function description

Short circuit detection callback.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **None:**

HAL_DFSDM_ChannelGetState

Function name

HAL_DFSDM_Channel_StateTypeDef HAL_DFSDM_ChannelGetState (DFSDM_Channel_HandleTypeDef * hdfsm_channel)

Function description

This function allows to get the current DFSDM channel handle state.

Parameters

- **hdfsm_channel:** : DFSDM channel handle.

Return values

- **DFSDM:** channel state.

HAL_DFSDM_FilterInit

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInit (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

Initialize the DFSDM filter according to the specified parameters in the DFSDM_FilterInitTypeDef structure and initialize the associated handle.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status.

HAL_DFSDM_FilterDeInit

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterDeInit (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

De-initializes the DFSDM filter.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status.

HAL_DFSDM_FilterMspInit

Function name

void HAL_DFSDM_FilterMspInit (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

Initializes the DFSDM filter MSP.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterMspDelInit

Function name

void HAL_DFSDM_FilterMspDelInit (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

De-initializes the DFSDM filter MSP.

Parameters

- **hdfsdm_filter:** : DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterConfigRegChannel

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterConfigRegChannel (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel, uint32_t ContinuousMode)

Function description

This function allows to select channel and to enable/disable continuous mode for regular conversion.

Parameters

- **hdfsdm_filter:** : DFSDM filter handle.
- **Channel:** : Channel for regular conversion. This parameter can be a value of DFSDM Channel Selection.
- **ContinuousMode:** : Enable/disable continuous mode for regular conversion. This parameter can be a value of DFSDM Continuous Mode.

Return values

- **HAL:** status

HAL_DFSDM_FilterConfigInjChannel

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterConfigInjChannel (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t Channel)

Function description

This function allows to select channels for injected conversion.

Parameters

- **hdfsdm_filter:** : DFSDM filter handle.
- **Channel:** : Channels for injected conversion. This parameter can be a values combination of DFSDM Channel Selection.

Return values

- **HAL:** status

HAL_DFSDM_FilterRegularStart

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to start regular conversion in polling mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing.

HAL_DFSDM_FilterRegularStart_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart_IT (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

This function allows to start regular conversion in interrupt mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing.

HAL_DFSDM_FilterRegularStart_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStart_DMA (DFSDM_Filter_HandleTypeDef * hdfsm_filter, int32_t * pData, uint32_t Length)

Function description

This function allows to start regular conversion in DMA mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.
- **pData:** : The destination buffer address.
- **Length:** : The length of data to be transferred from DFSDM filter to memory.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing. Please note that data on buffer will contain signed regular conversion value on 24 most significant bits and corresponding channel on 3 least significant bits.

HAL_DFSDM_FilterRegularMsbStart_DMA

Function name

```
HAL_StatusTypeDef HAL_DFSDM_FilterRegularMsbStart_DMA (DFSDM_Filter_HandleTypeDef *  
hdfsm_filter, int16_t * pData, uint32_t Length)
```

Function description

This function allows to start regular conversion in DMA mode and to get only the 16 most significant bits of conversion.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.
- **pData:** : The destination buffer address.
- **Length:** : The length of data to be transferred from DFSDM filter to memory.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if injected conversion is ongoing. Please note that data on buffer will contain signed 16 most significant bits of regular conversion.

HAL_DFSDM_FilterRegularStop

Function name

```
HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop (DFSDM_Filter_HandleTypeDef * hdfsm_filter)
```

Function description

This function allows to stop regular conversion in polling mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterRegularStop_IT

Function name

```
HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop_IT (DFSDM_Filter_HandleTypeDef * hdfsm_filter)
```

Function description

This function allows to stop regular conversion in interrupt mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterRegularStop_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterRegularStop_DMA (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

This function allows to stop regular conversion in DMA mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

This function allows to start injected conversion in polling mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart_IT (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

This function allows to start injected conversion in interrupt mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing.

HAL_DFSDM_FilterInjectedStart_DMA

Function name

```
HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStart_DMA (DFSDM_Filter_HandleTypeDef *  
hdfsm_filter, int32_t * pData, uint32_t Length)
```

Function description

This function allows to start injected conversion in DMA mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.
- **pData:** : The destination buffer address.
- **Length:** : The length of data to be transferred from DFSDM filter to memory.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing. Please note that data on buffer will contain signed injected conversion value on 24 most significant bits and corresponding channel on 3 least significant bits.

HAL_DFSDM_FilterInjectedMsbStart_DMA

Function name

```
HAL_StatusTypeDef HAL_DFSDM_FilterInjectedMsbStart_DMA (DFSDM_Filter_HandleTypeDef *  
hdfsm_filter, int16_t * pData, uint32_t Length)
```

Function description

This function allows to start injected conversion in DMA mode and to get only the 16 most significant bits of conversion.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.
- **pData:** : The destination buffer address.
- **Length:** : The length of data to be transferred from DFSDM filter to memory.

Return values

- **HAL:** status

Notes

- This function should be called only when DFSDM filter instance is in idle state or if regular conversion is ongoing. Please note that data on buffer will contain signed 16 most significant bits of injected conversion.

HAL_DFSDM_FilterInjectedStop

Function name

```
HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop (DFSDM_Filter_HandleTypeDef * hdfsm_filter)
```

Function description

This function allows to stop injected conversion in polling mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterInjectedStop_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop_IT (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

This function allows to stop injected conversion in interrupt mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterInjectedStop_DMA

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterInjectedStop_DMA (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

This function allows to stop injected conversion in DMA mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **HAL:** status

Notes

- This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterAwdStart_IT

Function name

HAL_StatusTypeDef HAL_DFSDM_FilterAwdStart_IT (DFSDM_Filter_HandleTypeDef * hdfsm_filter, DFSDM_Filter_AwdParamTypeDef * awdParam)

Function description

This function allows to start filter analog watchdog in interrupt mode.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.
- **awdParam:** : DFSDM filter analog watchdog parameters.

Return values

- **HAL:** status

HAL_DFSDM_FilterAwdStop_IT

Function name

`HAL_StatusTypeDef HAL_DFSDM_FilterAwdStop_IT (DFSDM_Filter_HandleTypeDef * hdfsm_filter)`

Function description

This function allows to stop filter analog watchdog in interrupt mode.

Parameters

- `hdfsm_filter`: DFSDM filter handle.

Return values

- `HAL`: status

HAL_DFSDM_FilterExdStart

Function name

`HAL_StatusTypeDef HAL_DFSDM_FilterExdStart (DFSDM_Filter_HandleTypeDef * hdfsm_filter, uint32_t Channel)`

Function description

This function allows to start extreme detector feature.

Parameters

- `hdfsm_filter`: DFSDM filter handle.
- `Channel`: Channels where extreme detector is enabled. This parameter can be a values combination of DFSDM Channel Selection.

Return values

- `HAL`: status

HAL_DFSDM_FilterExdStop

Function name

`HAL_StatusTypeDef HAL_DFSDM_FilterExdStop (DFSDM_Filter_HandleTypeDef * hdfsm_filter)`

Function description

This function allows to stop extreme detector feature.

Parameters

- `hdfsm_filter`: DFSDM filter handle.

Return values

- `HAL`: status

HAL_DFSDM_FilterGetRegularValue

Function name

`int32_t HAL_DFSDM_FilterGetRegularValue (DFSDM_Filter_HandleTypeDef * hdfsm_filter, uint32_t * Channel)`

Function description

This function allows to get regular conversion value.

Parameters

- `hdfsm_filter`: DFSDM filter handle.

- **Channel:** : Corresponding channel of regular conversion.

Return values

- **Regular:** conversion value

HAL_DFSDM_FilterGetInjectedValue

Function name

int32_t HAL_DFSDM_FilterGetInjectedValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)

Function description

This function allows to get injected conversion value.

Parameters

- **hdfsdm_filter:** : DFSDM filter handle.
- **Channel:** : Corresponding channel of injected conversion.

Return values

- **Injected:** conversion value

HAL_DFSDM_FilterGetExd.MaxValue

Function name

int32_t HAL_DFSDM_FilterGetExd.MaxValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)

Function description

This function allows to get extreme detector maximum value.

Parameters

- **hdfsdm_filter:** : DFSDM filter handle.
- **Channel:** : Corresponding channel.

Return values

- **Extreme:** detector maximum value This value is between Min_Data = -8388608 and Max_Data = 8388607.

HAL_DFSDM_FilterGetExd.MinValue

Function name

int32_t HAL_DFSDM_FilterGetExd.MinValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter, uint32_t * Channel)

Function description

This function allows to get extreme detector minimum value.

Parameters

- **hdfsdm_filter:** : DFSDM filter handle.
- **Channel:** : Corresponding channel.

Return values

- **Extreme:** detector minimum value This value is between Min_Data = -8388608 and Max_Data = 8388607.

HAL_DFSDM_FilterGetConvTimeValue

Function name

uint32_t HAL_DFSDM_FilterGetConvTimeValue (DFSDM_Filter_HandleTypeDef * hdfsdm_filter)

Function description

This function allows to get conversion time value.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **Conversion:** time value

Notes

- To get time in second, this value has to be divided by DFSDM clock frequency.

HAL_DFSDM_IRQHandler

Function name

```
void HAL_DFSDM_IRQHandler (DFSDM_Filter_HandleTypeDef * hdfsm_filter)
```

Function description

This function handles the DFSDM interrupts.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterPollForRegConversion

Function name

```
HAL_StatusTypeDef HAL_DFSDM_FilterPollForRegConversion (DFSDM_Filter_HandleTypeDef * hdfsm_filter, uint32_t Timeout)
```

Function description

This function allows to poll for the end of regular conversion.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.
- **Timeout:** : Timeout value in milliseconds.

Return values

- **HAL:** status

Notes

- This function should be called only if regular conversion is ongoing.

HAL_DFSDM_FilterPollForInjConversion

Function name

```
HAL_StatusTypeDef HAL_DFSDM_FilterPollForInjConversion (DFSDM_Filter_HandleTypeDef * hdfsm_filter, uint32_t Timeout)
```

Function description

This function allows to poll for the end of injected conversion.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.
- **Timeout:** : Timeout value in milliseconds.

Return values

- **HAL:** status

Notes

- This function should be called only if injected conversion is ongoing.

HAL_DFSDM_FilterRegConvCpltCallback

Function name

void HAL_DFSDM_FilterRegConvCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

Regular conversion complete callback.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **None:**

Notes

- In interrupt mode, user has to read conversion value in this function using **HAL_DFSDM_FilterGetRegularValue**.

HAL_DFSDM_FilterRegConvHalfCpltCallback

Function name

void HAL_DFSDM_FilterRegConvHalfCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

Half regular conversion complete callback.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterInjConvCpltCallback

Function name

void HAL_DFSDM_FilterInjConvCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsm_filter)

Function description

Injected conversion complete callback.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **None:**

Notes

- In interrupt mode, user has to read conversion value in this function using **HAL_DFSDM_FilterGetInjectedValue**.

HAL_DFSDM_FilterInjConvHalfCpltCallback

Function name

```
void HAL_DFSDM_FilterInjConvHalfCpltCallback (DFSDM_Filter_HandleTypeDef * hdfsm_filter)
```

Function description

Half injected conversion complete callback.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterAwdCallback

Function name

```
void HAL_DFSDM_FilterAwdCallback (DFSDM_Filter_HandleTypeDef * hdfsm_filter, uint32_t Channel,  
uint32_t Threshold)
```

Function description

Filter analog watchdog callback.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.
- **Channel:** : Corresponding channel.
- **Threshold:** : Low or high threshold has been reached.

Return values

- **None:**

HAL_DFSDM_FilterErrorCallback

Function name

```
void HAL_DFSDM_FilterErrorCallback (DFSDM_Filter_HandleTypeDef * hdfsm_filter)
```

Function description

Error callback.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **None:**

HAL_DFSDM_FilterGetState

Function name

```
HAL_DFSDM_Filter_StateTypeDef HAL_DFSDM_FilterGetState (DFSDM_Filter_HandleTypeDef *  
hdfsm_filter)
```

Function description

This function allows to get the current DFSDM filter handle state.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **DFSDM:** filter state.

HAL_DFSDM_FilterGetError

Function name

`uint32_t HAL_DFSDM_FilterGetError (DFSDM_Filter_HandleTypeDef * hdfsm_filter)`

Function description

This function allows to get the current DFSDM filter error.

Parameters

- **hdfsm_filter:** : DFSDM filter handle.

Return values

- **DFSDM:** filter error code.

17.3 DFSDM Firmware driver defines

17.3.1 DFSDM

DFSDM analog watchdog threshold

DFSDM_AWD_HIGH_THRESHOLD

Analog watchdog high threshold

DFSDM_AWD_LOW_THRESHOLD

Analog watchdog low threshold

DFSDM break signals

DFSDM_NO_BREAK_SIGNAL

No break signal

DFSDM_BREAK_SIGNAL_0

Break signal 0

DFSDM_BREAK_SIGNAL_1

Break signal 1

DFSDM_BREAK_SIGNAL_2

Break signal 2

DFSDM_BREAK_SIGNAL_3

Break signal 3

DFSDM channel analog watchdog filter order

DFSDM_CHANNEL_FASTSINC_ORDER

FastSinc filter type

DFSDM_CHANNEL_SINC1_ORDER

Sinc 1 filter type

DFSDM_CHANNEL_SINC2_ORDER

Sinc 2 filter type

DFSDM_CHANNEL_SINC3_ORDER

Sinc 3 filter type

DFSDM channel input data packing**DFSDM_CHANNEL_STANDARD_MODE**

Standard data packing mode

DFSDM_CHANNEL_INTERLEAVED_MODE

Interleaved data packing mode

DFSDM_CHANNEL_DUAL_MODE

Dual data packing mode

DFSDM channel input multiplexer**DFSDM_CHANNEL_EXTERNAL_INPUTS**

Data are taken from external inputs

DFSDM_CHANNEL_ADC_OUTPUT

Data are taken from ADC output

DFSDM_CHANNEL_INTERNAL_REGISTER

Data are taken from internal register

DFSDM channel input pins**DFSDM_CHANNEL_SAME_CHANNEL_PINS**

Input from pins on same channel

DFSDM_CHANNEL_FOLLOWING_CHANNEL_PINS

Input from pins on following channel

DFSDM channel output clock selection**DFSDM_CHANNEL_OUTPUT_CLOCK_SYSTEM**

Source for ouput clock is system clock

DFSDM_CHANNEL_OUTPUT_CLOCK_AUDIO

Source for ouput clock is audio clock

DFSDM Channel Selection**DFSDM_CHANNEL_0****DFSDM_CHANNEL_1****DFSDM_CHANNEL_2****DFSDM_CHANNEL_3****DFSDM_CHANNEL_4****DFSDM_CHANNEL_5****DFSDM_CHANNEL_6****DFSDM_CHANNEL_7**

DFSDM channel serial interface type**DFSDM_CHANNEL_SPI_RISING**

SPI with rising edge

DFSDM_CHANNEL_SPI_FALLING

SPI with falling edge

DFSDM_CHANNEL_MANCHESTER_RISING

Manchester with rising edge

DFSDM_CHANNEL_MANCHESTER_FALLING

Manchester with falling edge

DFSDM channel SPI clock selection**DFSDM_CHANNEL_SPI_CLOCK_EXTERNAL**

External SPI clock

DFSDM_CHANNEL_SPI_CLOCK_INTERNAL

Internal SPI clock

DFSDM_CHANNEL_SPI_CLOCK_INTERNAL_DIV2_FALLING

Internal SPI clock divided by 2, falling edge

DFSDM_CHANNEL_SPI_CLOCK_INTERNAL_DIV2_RISING

Internal SPI clock divided by 2, rising edge

DFSDM Continuous Mode**DFSDM_CONTINUOUS_CONV_OFF**

Conversion are not continuous

DFSDM_CONTINUOUS_CONV_ON

Conversion are continuous

DFSDM Exported Macros**_HAL_DFSDM_CHANNEL_RESET_HANDLE_STATE****Description:**

- Reset DFSDM channel handle state.

Parameters:

- **_HANDLE_**: DFSDM channel handle.

Return value:

- None

_HAL_DFSDM_FILTER_RESET_HANDLE_STATE**Description:**

- Reset DFSDM filter handle state.

Parameters:

- **_HANDLE_**: DFSDM filter handle.

Return value:

- None

DFSDM filter analog watchdog data source**DFSDM_FILTER_AWD_FILTER_DATA**

From digital filter

DFSDM_FILTER_AWD_CHANNEL_DATA

From analog watchdog channel

DFSDM filter error code**DFSDM_FILTER_ERROR_NONE**

No error

DFSDM_FILTER_ERROR_REGULAR_OVERRUN

OVERRUN occurs during regular conversion

DFSDM_FILTER_ERROR_INJECTED_OVERRUN

OVERRUN occurs during injected conversion

DFSDM_FILTER_ERROR_DMA

DMA error occurs

DFSDM filter external trigger**DFSDM_FILTER_EXT_TRIG_TIM1_TRGO**

For DFSDM 0, 1, 2 and 3

DFSDM_FILTER_EXT_TRIG_TIM1_TRGO2

For DFSDM 0, 1, 2 and 3

DFSDM_FILTER_EXT_TRIG_TIM8_TRGO

For DFSDM 0, 1, 2 and 3

DFSDM_FILTER_EXT_TRIG_TIM8_TRGO2

For DFSDM 0, 1 and 2

DFSDM_FILTER_EXT_TRIG_TIM3_TRGO

For DFSDM 3

DFSDM_FILTER_EXT_TRIG_TIM4_TRGO

For DFSDM 0, 1 and 2

DFSDM_FILTER_EXT_TRIG_TIM16_OC1

For DFSDM 3

DFSDM_FILTER_EXT_TRIG_TIM6_TRGO

For DFSDM 0 and 1

DFSDM_FILTER_EXT_TRIG_TIM7_TRGO

For DFSDM 2 and 3

DFSDM_FILTER_EXT_TRIG_HRTIM1_ADCTRG1**DFSDM_FILTER_EXT_TRIG_HRTIM1_ADCTRG3****DFSDM_FILTER_EXT_TRIG EXTI11**

For DFSDM 0, 1, 2 and 3

DFSDM_FILTER_EXT_TRIG EXTI15

For DFSDM 0, 1, 2 and 3

DFSDM_FILTER_EXT_TRIG_LPTIM1

For DFSDM 0, 1, 2 and 3

DFSDM_FILTER_EXT_TRIG_LPTIM2

For DFSDM 0, 1, 2 and 3

DFSDM_FILTER_EXT_TRIG_LPTIM3

For DFSDM 0, 1, 2 and 3

DFSDM filter external trigger edge

DFSDM_FILTER_EXT_TRIG_RISING_EDGE

External rising edge

DFSDM_FILTER_EXT_TRIG_FALLING_EDGE

External falling edge

DFSDM_FILTER_EXT_TRIG_BOTH_EDGES

External rising and falling edges

DFSDM filter sinc order

DFSDM_FILTER_FASTSINC_ORDER

FastSinc filter type

DFSDM_FILTER_SINC1_ORDER

Sinc 1 filter type

DFSDM_FILTER_SINC2_ORDER

Sinc 2 filter type

DFSDM_FILTER_SINC3_ORDER

Sinc 3 filter type

DFSDM_FILTER_SINC4_ORDER

Sinc 4 filter type

DFSDM_FILTER_SINC5_ORDER

Sinc 5 filter type

DFSDM filter conversion trigger

DFSDM_FILTER_SW_TRIGGER

Software trigger

DFSDM_FILTER_SYNC_TRIGGER

Synchronous with DFSDM0

DFSDM_FILTER_EXT_TRIGGER

External trigger (only for injected conversion)

18 HAL DMA2D Generic Driver

18.1 DMA2D Firmware driver registers structures

18.1.1 DMA2D_ColorTypeDef

Data Fields

- *uint32_t Blue*
- *uint32_t Green*
- *uint32_t Red*

Field Documentation

- *uint32_t DMA2D_ColorTypeDef::Blue*

Configures the blue value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

- *uint32_t DMA2D_ColorTypeDef::Green*

Configures the green value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

- *uint32_t DMA2D_ColorTypeDef::Red*

Configures the red value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

18.1.2 DMA2D_CLUTCfgTypeDef

Data Fields

- *uint32_t * pCLUT*
- *uint32_t CLUTColorMode*
- *uint32_t Size*

Field Documentation

- *uint32_t* DMA2D_CLUTCfgTypeDef::pCLUT*

Configures the DMA2D CLUT memory address.

- *uint32_t DMA2D_CLUTCfgTypeDef::CLUTColorMode*

Configures the DMA2D CLUT color mode. This parameter can be one value of **DMA2D CLUT Color Mode**.

- *uint32_t DMA2D_CLUTCfgTypeDef::Size*

Configures the DMA2D CLUT size. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

18.1.3 DMA2D_InitTypeDef

Data Fields

- *uint32_t Mode*
- *uint32_t ColorMode*
- *uint32_t OutputOffset*
- *uint32_t AlphaInverted*
- *uint32_t RedBlueSwap*

Field Documentation

- **`uint32_t DMA2D_InitTypeDef::Mode`**
Configures the DMA2D transfer mode. This parameter can be one value of **DMA2D Mode**.
- **`uint32_t DMA2D_InitTypeDef::ColorMode`**
Configures the color format of the output image. This parameter can be one value of **DMA2D Output Color Mode**.
- **`uint32_t DMA2D_InitTypeDef::OutputOffset`**
Specifies the Offset value. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- **`uint32_t DMA2D_InitTypeDef::AlphaInverted`**
Select regular or inverted alpha value for the output pixel format converter. This parameter can be one value of **DMA2D ALPHA Inversion**.
- **`uint32_t DMA2D_InitTypeDef::RedBlueSwap`**
Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR) for the output pixel format converter. This parameter can be one value of **DMA2D Red and Blue Swap**.

18.1.4 DMA2D_LayerCfgTypeDef

Data Fields

- **`uint32_t InputOffset`**
- **`uint32_t InputColorMode`**
- **`uint32_t AlphaMode`**
- **`uint32_t InputAlpha`**
- **`uint32_t AlphaInverted`**
- **`uint32_t RedBlueSwap`**
- **`uint32_t ChromaSubSampling`**

Field Documentation

- **`uint32_t DMA2D_LayerCfgTypeDef::InputOffset`**
Configures the DMA2D foreground or background offset. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- **`uint32_t DMA2D_LayerCfgTypeDef::InputColorMode`**
Configures the DMA2D foreground or background color mode. This parameter can be one value of **DMA2D Input Color Mode**.
- **`uint32_t DMA2D_LayerCfgTypeDef::AlphaMode`**
Configures the DMA2D foreground or background alpha mode. This parameter can be one value of **DMA2D Alpha Mode**.
- **`uint32_t DMA2D_LayerCfgTypeDef::InputAlpha`**
Specifies the DMA2D foreground or background alpha value and color value in case of A8 or A4 color mode. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF except for the color modes detailed below.

Note:

- In case of A8 or A4 color mode (ARGB), this parameter must be a number between Min_Data = 0x00000000 and Max_Data = 0xFFFFFFFF where
 - InputAlpha[24:31] is the alpha value ALPHA[0:7]
 - InputAlpha[16:23] is the red value RED[0:7]
 - InputAlpha[8:15] is the green value GREEN[0:7]

- InputAlpha[0:7] is the blue value BLUE[0:7].
- **`uint32_t DMA2D_LayerCfgTypeDef::AlphaInverted`**
Select regular or inverted alpha value. This parameter can be one value of **DMA2D ALPHA Inversion**.
- **`uint32_t DMA2D_LayerCfgTypeDef::RedBlueSwap`**
Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR). This parameter can be one value of **DMA2D Red and Blue Swap**.
- **`uint32_t DMA2D_LayerCfgTypeDef::ChromaSubSampling`**
Configure the chroma sub-sampling mode for the YCbCr color mode This parameter can be one value of **DMA2D Chroma Sub Sampling**

18.1.5 **`__DMA2D_HandleTypeDef`**

Data Fields

- **`DMA2D_TypeDef * Instance`**
- **`DMA2D_InitTypeDef Init`**
- **`void(* XferCpltCallback`**
- **`void(* XferErrorCallback`**
- **`DMA2D_LayerCfgTypeDef LayerCfg`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_DMA2D_StateTypeDef State`**
- **`__IO uint32_t ErrorCode`**

Field Documentation

- **`DMA2D_TypeDef* __DMA2D_HandleTypeDef::Instance`**
DMA2D register base address.
- **`DMA2D_InitTypeDef __DMA2D_HandleTypeDef::Init`**
DMA2D communication parameters.
- **`void(* __DMA2D_HandleTypeDef::XferCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)`**
DMA2D transfer complete callback.
- **`void(* __DMA2D_HandleTypeDef::XferErrorCallback)(struct __DMA2D_HandleTypeDef *hdma2d)`**
DMA2D transfer error callback.
- **`DMA2D_LayerCfgTypeDef __DMA2D_HandleTypeDef::LayerCfg[MAX_DMA2D_LAYER]`**
DMA2D Layers parameters
- **`HAL_LockTypeDef __DMA2D_HandleTypeDef::Lock`**
DMA2D lock.
- **`__IO HAL_DMA2D_StateTypeDef __DMA2D_HandleTypeDef::State`**
DMA2D transfer state.
- **`__IO uint32_t __DMA2D_HandleTypeDef::ErrorCode`**
DMA2D error code.

18.2 DMA2D Firmware driver API description

18.2.1 How to use this driver

1. Program the required configuration through the following parameters: the transfer mode, the output color mode and the output offset using `HAL_DMA2D_Init()` function.

2. Program the required configuration through the following parameters: the input color mode, the input color, the input alpha value, the alpha mode, the red/blue swap mode, the inverted alpha mode and the input offset using HAL_DMA2D_ConfigLayer() function for foreground or/and background layer.

Polling mode IO operation

1. Configure pdata parameter (explained hereafter), destination and data length and enable the transfer using HAL_DMA2D_Start().
2. Wait for end of transfer using HAL_DMA2D_PollForTransfer(), at this stage user can specify the value of timeout according to his end application.

Interrupt mode IO operation

1. Configure pdata parameter, destination and data length and enable the transfer using HAL_DMA2D_Start_IT().
2. Use HAL_DMA2D_IRQHandler() called under DMA2D_IRQHandler() interrupt subroutine.
3. At the end of data transfer HAL_DMA2D_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback (member of DMA2D handle structure).
4. In case of error, the HAL_DMA2D_IRQHandler() function will call the callback XferErrorCallback.

Note: *In Register-to-Memory transfer mode, pdata parameter is the register color, in Memory-to-memory or Memory-to-Memory with pixel format conversion pdata is the source address.*

Note: *Configure the foreground source address, the background source address, the destination and data length then Enable the transfer using HAL_DMA2D_BlendingStart() in polling mode and HAL_DMA2D_BlendingStart_IT() in interrupt mode.*

Note: *HAL_DMA2D_BlendingStart() and HAL_DMA2D_BlendingStart_IT() functions are used if the memory to memory with blending transfer mode is selected.*

5. Optionally, configure and enable the CLUT using HAL_DMA2D_CLUTLoad() in polling mode or HAL_DMA2D_CLUTLoad_IT() in interrupt mode.
6. Optionally, configure the line watermark in using the API HAL_DMA2D_ProgramLineEvent()
7. Optionally, configure the dead time value in the AHB clock cycle inserted between two consecutive accesses on the AHB master port in using the API HAL_DMA2D_ConfigDeadTime() and enable/disable the functionality with the APIs HAL_DMA2D_EnableDeadTime() or HAL_DMA2D_DisableDeadTime().
8. The transfer can be suspended, resumed and aborted using the following functions:
HAL_DMA2D_Suspend(), HAL_DMA2D_Resume(), HAL_DMA2D_Abort().
9. The CLUT loading can be suspended, resumed and aborted using the following functions:
HAL_DMA2D_CLUTLoading_Suspend(), HAL_DMA2D_CLUTLoading_Resume(),
HAL_DMA2D_CLUTLoading_Abort().
10. To control the DMA2D state, use the following function: HAL_DMA2D_GetState().
11. To read the DMA2D error code, use the following function: HAL_DMA2D_GetError().

DMA2D HAL driver macros list

Below the list of most used macros in DMA2D HAL driver :

- __HAL_DMA2D_ENABLE: Enable the DMA2D peripheral.
- __HAL_DMA2D_GET_FLAG: Get the DMA2D pending flags.
- __HAL_DMA2D_CLEAR_FLAG: Clear the DMA2D pending flags.
- __HAL_DMA2D_ENABLE_IT: Enable the specified DMA2D interrupts.
- __HAL_DMA2D_DISABLE_IT: Disable the specified DMA2D interrupts.
- __HAL_DMA2D_GET_IT_SOURCE: Check whether the specified DMA2D interrupt is enabled or not.

Note: *You can refer to the DMA2D HAL driver header file for more useful macros*

18.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DMA2D

- De-initialize the DMA2D

This section contains the following APIs:

- [*HAL_DMA2D_Init*](#)
- [*HAL_DMA2D_DelInit*](#)
- [*HAL_DMA2D_MspInit*](#)
- [*HAL_DMA2D_MspDelInit*](#)

18.2.3 IO operation functions

This section provides functions allowing to:

- Configure the pdata, destination address and data size then start the DMA2D transfer.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer.
- Configure the pdata, destination address and data size then start the DMA2D transfer with interrupt.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer with interrupt.
- Abort DMA2D transfer.
- Suspend DMA2D transfer.
- Resume DMA2D transfer.
- Enable CLUT transfer.
- Configure CLUT loading then start transfer in polling mode.
- Configure CLUT loading then start transfer in interrupt mode.
- Abort DMA2D CLUT loading.
- Suspend DMA2D CLUT loading.
- Resume DMA2D CLUT loading.
- Poll for transfer complete.
- handle DMA2D interrupt request.
- Transfer watermark callback.
- CLUT Transfer Complete callback.

This section contains the following APIs:

- [*HAL_DMA2D_Start*](#)
- [*HAL_DMA2D_Start_IT*](#)
- [*HAL_DMA2D_BlendingStart*](#)
- [*HAL_DMA2D_BlendingStart_IT*](#)
- [*HAL_DMA2D_Abort*](#)
- [*HAL_DMA2D_Suspend*](#)
- [*HAL_DMA2D_Resume*](#)
- [*HAL_DMA2D_EnableCLUT*](#)
- [*HAL_DMA2D_CLUTLoad*](#)
- [*HAL_DMA2D_CLUTLoad_IT*](#)
- [*HAL_DMA2D_CLUTLoading_Abort*](#)
- [*HAL_DMA2D_CLUTLoading_Suspend*](#)
- [*HAL_DMA2D_CLUTLoading_Resume*](#)
- [*HAL_DMA2D_PollForTransfer*](#)
- [*HAL_DMA2D_IRQHandler*](#)
- [*HAL_DMA2D_LineEventCallback*](#)
- [*HAL_DMA2D_CLUTLoadingCpltCallback*](#)

18.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the DMA2D foreground or background layer parameters.
- Configure the DMA2D CLUT transfer.
- Configure the line watermark
- Configure the dead time value.
- Enable or disable the dead time value functionality.

This section contains the following APIs:

- [*HAL_DMA2D_ConfigLayer*](#)
- [*HAL_DMA2D_ConfigCLUT*](#)
- [*HAL_DMA2D_ProgramLineEvent*](#)
- [*HAL_DMA2D_EnableDeadTime*](#)
- [*HAL_DMA2D_DisableDeadTime*](#)
- [*HAL_DMA2D_ConfigDeadTime*](#)

18.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to :

- Get the DMA2D state
- Get the DMA2D error code

This section contains the following APIs:

- [*HAL_DMA2D_GetState*](#)
- [*HAL_DMA2D_GetError*](#)

18.2.6 Detailed description of functions

[**HAL_DMA2D_Init**](#)

Function name

HAL_StatusTypeDef HAL_DMA2D_Init (DMA2D_HandleTypeDef * hdma2d)

Function description

Initialize the DMA2D according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

[**HAL_DMA2D_DeInit**](#)

Function name

HAL_StatusTypeDef HAL_DMA2D_DeInit (DMA2D_HandleTypeDef * hdma2d)

Function description

Deinitializes the DMA2D peripheral registers to their default reset values.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_MspInit

Function name

void HAL_DMA2D_MspInit (DMA2D_HandleTypeDef * hdma2d)

Function description

Initializes the DMA2D MSP.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_MspDeInit

Function name

void HAL_DMA2D_MspDeInit (DMA2D_HandleTypeDef * hdma2d)

Function description

DeInitializes the DMA2D MSP.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_Start

Function name

HAL_StatusTypeDef HAL_DMA2D_Start (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the DMA2D Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata:** Configure the source memory Buffer address if Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_BlendingStart

Function name

HAL_StatusTypeDef HAL_DMA2D_BlendingStart (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the multi-source DMA2D Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1:** The source memory Buffer address for the foreground layer.
- **SrcAddress2:** The source memory Buffer address for the background layer.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_Start_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the DMA2D Transfer with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata:** Configure the source memory Buffer address if the Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_BlendingStart_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_BlendingStart_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the multi-source DMA2D Transfer with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1:** The source memory Buffer address for the foreground layer.
- **SrcAddress2:** The source memory Buffer address for the background layer.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_Suspend

Function name

HAL_StatusTypeDef HAL_DMA2D_Suspend (DMA2D_HandleTypeDef * hdma2d)

Function description

Suspend the DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_Resume

Function name

HAL_StatusTypeDef HAL_DMA2D_Resume (DMA2D_HandleTypeDef * hdma2d)

Function description

Resume the DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_Abort

Function name

HAL_StatusTypeDef HAL_DMA2D_Abort (DMA2D_HandleTypeDef * hdma2d)

Function description

Abort the DMA2D Transfer.

Parameters

- **hdma2d:** : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_EnableCLUT

Function name

HAL_StatusTypeDef HAL_DMA2D_EnableCLUT (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Enable the DMA2D CLUT Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoad

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoad (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **HAL:** status

Notes

- Invoking this API is similar to calling HAL_DMA2D_ConfigCLUT() then HAL_DMA2D_EnableCLUT().

HAL_DMA2D_CLUTLoad_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoad_IT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.

- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoading_Abort

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Abort (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Abort the DMA2D CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoading_Suspend

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Suspend (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Suspend the DMA2D CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoading_Resume

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Resume (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Resume the DMA2D CLUT loading.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **HAL:** status

HAL_DMA2D_PollForTransfer**Function name**

HAL_StatusTypeDef HAL_DMA2D_PollForTransfer (DMA2D_HandleTypeDef * hdma2d, uint32_t Timeout)

Function description

Polling for transfer complete or CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_DMA2D_IRQHandler**Function name**

void HAL_DMA2D_IRQHandler (DMA2D_HandleTypeDef * hdma2d)

Function description

Handle DMA2D interrupt request.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_LineEventCallback**Function name**

void HAL_DMA2D_LineEventCallback (DMA2D_HandleTypeDef * hdma2d)

Function description

Transfer watermark callback.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_CLUTLoadingCpltCallback**Function name**

void HAL_DMA2D_CLUTLoadingCpltCallback (DMA2D_HandleTypeDef * hdma2d)

Function description

CLUT Transfer Complete callback.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_ConfigLayer

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigLayer (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Configure the DMA2D Layer according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **HAL:** status

HAL_DMA2D_ConfigCLUT

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigCLUT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Configure the DMA2D CLUT Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **HAL:** status

HAL_DMA2D_ProgramLineEvent

Function name

HAL_StatusTypeDef HAL_DMA2D_ProgramLineEvent (DMA2D_HandleTypeDef * hdma2d, uint32_t Line)

Function description

Configure the line watermark.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **Line:** Line Watermark configuration (maximum 16-bit long value expected).

Return values

- **HAL:** status

Notes

- HAL_DMA2D_ProgramLineEvent() API enables the transfer watermark interrupt.
- The transfer watermark interrupt is disabled once it has occurred.

HAL_DMA2D_EnableDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_EnableDeadTime (DMA2D_HandleTypeDef * hdma2d)

Function description

Enable DMA2D dead time feature.

Parameters

- **hdma2d:** DMA2D handle.

Return values

- **HAL:** status

HAL_DMA2D_DisableDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_DisableDeadTime (DMA2D_HandleTypeDef * hdma2d)

Function description

Disable DMA2D dead time feature.

Parameters

- **hdma2d:** DMA2D handle.

Return values

- **HAL:** status

HAL_DMA2D_ConfigDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigDeadTime (DMA2D_HandleTypeDef * hdma2d, uint8_t DeadTime)

Function description

Configure dead time.

Parameters

- **hdma2d:** DMA2D handle.
- **DeadTime:** dead time value.

Return values

- **HAL:** status

Notes

- The dead time value represents the guaranteed minimum number of cycles between two consecutive transactions on the AHB bus.

HAL_DMA2D_GetState

Function name

HAL_DMA2D_StateTypeDef HAL_DMA2D_GetState (DMA2D_HandleTypeDef * hdma2d)

Function description

Return the DMA2D state.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** state

HAL_DMA2D_GetError

Function name

uint32_t HAL_DMA2D_GetError (DMA2D_HandleTypeDef * hdma2d)

Function description

Return the DMA2D error code.

Parameters

- **hdma2d:** : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for DMA2D.

Return values

- **DMA2D:** Error Code

18.3 DMA2D Firmware driver defines

18.3.1 DMA2D

DMA2D API Aliases

HAL_DMA2D_DisableCLUT

Aliased to HAL_DMA2D_CLUTLoading_Abort for compatibility with legacy code

DMA2D ALPHA Inversion

DMA2D_REGULAR_ALPHA

No modification of the alpha channel value

DMA2D_INVERTED_ALPHA

Invert the alpha channel value

DMA2D Alpha Mode

DMA2D_NO_MODIF_ALPHA

No modification of the alpha channel value

DMA2D_REPLACE_ALPHA

Replace original alpha channel value by programmed alpha value

DMA2D_COMBINE_ALPHA

Replace original alpha channel value by programmed alpha value with original alpha channel value

DMA2D Chroma Sub Sampling**DMA2D_NO_CSS**

No chroma sub-sampling 4:4:4

DMA2D_CSS_422

chroma sub-sampling 4:2:2

DMA2D_CSS_420

chroma sub-sampling 4:2:0

DMA2D CLUT Color Mode**DMA2D_CCM_ARGB8888**

ARGB8888 DMA2D CLUT color mode

DMA2D_CCM_RGB888

RGB888 DMA2D CLUT color mode

DMA2D CLUT Size**DMA2D_CLUT_SIZE**

DMA2D maximum CLUT size

DMA2D Color Value**DMA2D_COLOR_VALUE**

Color value mask

DMA2D Error Code**HAL_DMA2D_ERROR_NONE**

No error

HAL_DMA2D_ERROR_TE

Transfer error

HAL_DMA2D_ERROR_CE

Configuration error

HAL_DMA2D_ERROR_CAE

CLUT access error

HAL_DMA2D_ERROR_TIMEOUT

Timeout error

DMA2D Exported Macros**_HAL_DMA2D_RESET_HANDLE_STATE****Description:**

- Reset DMA2D handle state.

Parameters:

- **_HANDLE_**: specifies the DMA2D handle.

Return value:

- None

[__HAL_DMA2D_ENABLE](#)**Description:**

- Enable the DMA2D.

Parameters:

- `__HANDLE__`: DMA2D handle

Return value:

- None.

[__HAL_DMA2D_GET_FLAG](#)**Description:**

- Get the DMA2D pending flags.

Parameters:

- `__HANDLE__`: DMA2D handle
- `__FLAG__`: flag to check. This parameter can be any combination of the following values:
 - `DMA2D_FLAG_CE`: Configuration error flag
 - `DMA2D_FLAG_CTC`: CLUT transfer complete flag
 - `DMA2D_FLAG_CAE`: CLUT access error flag
 - `DMA2D_FLAG_TW`: Transfer Watermark flag
 - `DMA2D_FLAG_TC`: Transfer complete flag
 - `DMA2D_FLAG_TE`: Transfer error flag

Return value:

- The: state of FLAG.

[__HAL_DMA2D_CLEAR_FLAG](#)**Description:**

- Clear the DMA2D pending flags.

Parameters:

- `__HANDLE__`: DMA2D handle
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
 - `DMA2D_FLAG_CE`: Configuration error flag
 - `DMA2D_FLAG_CTC`: CLUT transfer complete flag
 - `DMA2D_FLAG_CAE`: CLUT access error flag
 - `DMA2D_FLAG_TW`: Transfer Watermark flag
 - `DMA2D_FLAG_TC`: Transfer complete flag
 - `DMA2D_FLAG_TE`: Transfer error flag

Return value:

- None

[__HAL_DMA2D_ENABLE_IT](#)**Description:**

- Enable the specified DMA2D interrupts.

Parameters:

- `__HANDLE__`: DMA2D handle

- `__INTERRUPT__`: specifies the DMA2D interrupt sources to be enabled. This parameter can be any combination of the following values:
 - DMA2D_IT_CE: Configuration error interrupt mask
 - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
 - DMA2D_IT_CAE: CLUT access error interrupt mask
 - DMA2D_IT_TW: Transfer Watermark interrupt mask
 - DMA2D_IT_TC: Transfer complete interrupt mask
 - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA2D_DISABLE_IT**Description:**

- Disable the specified DMA2D interrupts.

Parameters:

- `__HANDLE__`: DMA2D handle
- `__INTERRUPT__`: specifies the DMA2D interrupt sources to be disabled. This parameter can be any combination of the following values:
 - DMA2D_IT_CE: Configuration error interrupt mask
 - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
 - DMA2D_IT_CAE: CLUT access error interrupt mask
 - DMA2D_IT_TW: Transfer Watermark interrupt mask
 - DMA2D_IT_TC: Transfer complete interrupt mask
 - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA2D_GET_IT_SOURCE**Description:**

- Check whether the specified DMA2D interrupt source is enabled or not.

Parameters:

- `__HANDLE__`: DMA2D handle
- `__INTERRUPT__`: specifies the DMA2D interrupt source to check. This parameter can be one of the following values:
 - DMA2D_IT_CE: Configuration error interrupt mask
 - DMA2D_IT_CTC: CLUT transfer complete interrupt mask
 - DMA2D_IT_CAE: CLUT access error interrupt mask
 - DMA2D_IT_TW: Transfer Watermark interrupt mask
 - DMA2D_IT_TC: Transfer complete interrupt mask
 - DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- The: state of INTERRUPT source.

DMA2D Exported Types**MAX_DMA2D_LAYER**

DMA2D maximum number of layers

DMA2D Flags

DMA2D_FLAG_CE

Configuration Error Interrupt Flag

DMA2D_FLAG_CTC

CLUT Transfer Complete Interrupt Flag

DMA2D_FLAG_CAE

CLUT Access Error Interrupt Flag

DMA2D_FLAG_TW

Transfer Watermark Interrupt Flag

DMA2D_FLAG_TC

Transfer Complete Interrupt Flag

DMA2D_FLAG_TE

Transfer Error Interrupt Flag

DMA2D Input Color Mode**DMA2D_INPUT_ARGB8888**

ARGB8888 color mode

DMA2D_INPUT_RGB888

RGB888 color mode

DMA2D_INPUT_RGB565

RGB565 color mode

DMA2D_INPUT_ARGB1555

ARGB1555 color mode

DMA2D_INPUT_ARGB4444

ARGB4444 color mode

DMA2D_INPUT_L8

L8 color mode

DMA2D_INPUT_AL44

AL44 color mode

DMA2D_INPUT_AL88

AL88 color mode

DMA2D_INPUT_L4

L4 color mode

DMA2D_INPUT_A8

A8 color mode

DMA2D_INPUT_A4

A4 color mode

DMA2D_INPUT_YCBCR

YCbCr color mode

DMA2D Interrupts

DMA2D_IT_CE

Configuration Error Interrupt

DMA2D_IT_CTC

CLUT Transfer Complete Interrupt

DMA2D_IT_CAE

CLUT Access Error Interrupt

DMA2D_IT_TW

Transfer Watermark Interrupt

DMA2D_IT_TC

Transfer Complete Interrupt

DMA2D_IT_TE

Transfer Error Interrupt

DMA2D Maximum Line Watermark**DMA2D_LINE_WATERMARK_MAX**

DMA2D maximum line watermark

DMA2D Mode**DMA2D_M2M**

DMA2D memory to memory transfer mode

DMA2D_M2M_PFC

DMA2D memory to memory with pixel format conversion transfer mode

DMA2D_M2M_BLEND

DMA2D memory to memory with blending transfer mode

DMA2D_R2M

DMA2D register to memory transfer mode

DMA2D Offset**DMA2D_OFFSET**

maximum Line Offset

DMA2D Output Color Mode**DMA2D_OUTPUT_ARGB8888**

ARGB8888 DMA2D color mode

DMA2D_OUTPUT_RGB888

RGB888 DMA2D color mode

DMA2D_OUTPUT_RGB565

RGB565 DMA2D color mode

DMA2D_OUTPUT_ARGB1555

ARGB1555 DMA2D color mode

DMA2D_OUTPUT_ARGB4444

ARGB4444 DMA2D color mode

DMA2D Red and Blue Swap**DMA2D_RB_REGULAR**

Select regular mode (RGB or ARGB)

DMA2D_RB_SWAP

Select swap mode (BGR or ABGR)

DMA2D Shifts**DMA2D_POSITION_FGPCCR_CS**

Required left shift to set foreground CLUT size

DMA2D_POSITION_BGPCCR_CS

Required left shift to set background CLUT size

DMA2D_POSITION_FGPCCR_CCM

Required left shift to set foreground CLUT color mode

DMA2D_POSITION_BGPCCR_CCM

Required left shift to set background CLUT color mode

DMA2D_POSITION_OPFCCR_AI

Required left shift to set output alpha inversion

DMA2D_POSITION_FGPCCR_AI

Required left shift to set foreground alpha inversion

DMA2D_POSITION_BGPCCR_AI

Required left shift to set background alpha inversion

DMA2D_POSITION_OPFCCR_RBS

Required left shift to set output Red/Blue swap

DMA2D_POSITION_FGPCCR_RBS

Required left shift to set foreground Red/Blue swap

DMA2D_POSITION_BGPCCR_RBS

Required left shift to set background Red/Blue swap

DMA2D_POSITION_AMTCR_DT

Required left shift to set deadtime value

DMA2D_POSITION_FGPCCR_AM

Required left shift to set foreground alpha mode

DMA2D_POSITION_BGPCCR_AM

Required left shift to set background alpha mode

DMA2D_POSITION_FGPCCR_ALPHA

Required left shift to set foreground alpha value

DMA2D_POSITION_BGPCCR_ALPHA

Required left shift to set background alpha value

DMA2D_POSITION_NLR_PL

Required left shift to set pixels per lines value

DMA2D_POSITION_FGPFCCR_CSS

Required left shift to set foreground Chroma sub-sampling

DMA2D Size**DMA2D_PIXEL**

DMA2D maximum number of pixels per line

DMA2D_LINE

DMA2D maximum number of lines

DMA2D Time Out**DMA2D_TIMEOUT_ABORT**

1s

DMA2D_TIMEOUT_SUSPEND

1s

19 HAL DMA Generic Driver

19.1 DMA Firmware driver registers structures

19.1.1 DMA_InitTypeDef

Data Fields

- `uint32_t Request`
- `uint32_t Direction`
- `uint32_t PeriphInc`
- `uint32_t MemInc`
- `uint32_t PeriphDataAlignment`
- `uint32_t MemDataAlignment`
- `uint32_t Mode`
- `uint32_t Priority`
- `uint32_t FIFOMode`
- `uint32_t FIFOThreshold`
- `uint32_t MemBurst`
- `uint32_t PeriphBurst`

Field Documentation

- `uint32_t DMA_InitTypeDef::Request`

Specifies the request selected for the specified stream. This parameter can be a value of **DMA Request selection**

- `uint32_t DMA_InitTypeDef::Direction`

Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of **DMA Data transfer direction**

- `uint32_t DMA_InitTypeDef::PeriphInc`

Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of **DMA Peripheral incremented mode**

- `uint32_t DMA_InitTypeDef::MemInc`

Specifies whether the memory address register should be incremented or not. This parameter can be a value of **DMA Memory incremented mode**

- `uint32_t DMA_InitTypeDef::PeriphDataAlignment`

Specifies the Peripheral data width. This parameter can be a value of **DMA Peripheral data size**

- `uint32_t DMA_InitTypeDef::MemDataAlignment`

Specifies the Memory data width. This parameter can be a value of **DMA Memory data size**

- `uint32_t DMA_InitTypeDef::Mode`

Specifies the operation mode of the DMAy Streamx. This parameter can be a value of **DMA mode**

Note:

- The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Stream
- `uint32_t DMA_InitTypeDef::Priority`

Specifies the software priority for the DMAy Streamx. This parameter can be a value of **DMA Priority level**

- `uint32_t DMA_InitTypeDef::FIFOMode`

Specifies if the FIFO mode or Direct mode will be used for the specified stream. This parameter can be a value of **DMA FIFO direct mode**

Note:

- The Direct mode (FIFO mode disabled) cannot be used if the memory-to-memory data transfer is configured on the selected stream

- `uint32_t DMA_InitTypeDef::FIFOThreshold`

Specifies the FIFO threshold level. This parameter can be a value of **DMA FIFO threshold level**

- `uint32_t DMA_InitTypeDef::MemBurst`

Specifies the Burst transfer configuration for the memory transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of **DMA Memory burst**

Note:

- The burst mode is possible only if the address Increment mode is enabled.

- `uint32_t DMA_InitTypeDef::PeriphBurst`

Specifies the Burst transfer configuration for the peripheral transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of **DMA Peripheral burst**

Note:

- The burst mode is possible only if the address Increment mode is enabled.

19.1.2 __DMA_HandleTypeDef

Data Fields

- `void * Instance`
- `DMA_InitTypeDef Init`
- `HAL_LockTypeDef Lock`
- `__IO HAL_DMA_StateTypeDef State`
- `void * Parent`
- `void(* XferCpltCallback`
- `void(* XferHalfCpltCallback`
- `void(* XferM1CpltCallback`
- `void(* XferM1HalfCpltCallback`
- `void(* XferErrorCallback`
- `void(* XferAbortCallback`
- `__IO uint32_t ErrorCode`
- `uint32_t StreamBaseAddress`
- `uint32_t StreamIndex`
- `DMAMUX_Channel_TypeDef * DMAMuxChannel`
- `DMAMUX_ChannelStatus_TypeDef * DMAMuxChannelStatus`
- `uint32_t DMAMuxChannelStatusMask`
- `DMAMUX_RequestGen_TypeDef * DMAMuxRequestGen`
- `DMAMUX_RequestGenStatus_TypeDef * DMAMuxRequestGenStatus`
- `uint32_t DMAMuxRequestGenStatusMask`

Field Documentation

- `void* __DMA_HandleTypeDef::Instance`
Register base address
- `DMA_InitTypeDef __DMA_HandleTypeDef::Init`

- DMA communication parameters
- **`HAL_LockTypeDef __DMA_HandleTypeDef::Lock`**
DMA locking object
- **`_IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State`**
DMA transfer state
- **`void* __DMA_HandleTypeDef::Parent`**
Parent object state
- **`void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)`**
DMA transfer complete callback
- **`void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)`**
DMA Half transfer complete callback
- **`void(* __DMA_HandleTypeDef::XferM1CpltCallback)(struct __DMA_HandleTypeDef *hdma)`**
DMA transfer complete Memory1 callback
- **`void(* __DMA_HandleTypeDef::XferM1HalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)`**
DMA transfer Half complete Memory1 callback
- **`void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)`**
DMA transfer error callback
- **`void(* __DMA_HandleTypeDef::XferAbortCallback)(struct __DMA_HandleTypeDef *hdma)`**
DMA transfer Abort callback
- **`_IO uint32_t __DMA_HandleTypeDef::ErrorCode`**
DMA Error code
- **`uint32_t __DMA_HandleTypeDef::StreamBaseAddress`**
DMA Stream Base Address
- **`uint32_t __DMA_HandleTypeDef::StreamIndex`**
DMA Stream Index
- **`DMAMUX_Channel_TypeDef* __DMA_HandleTypeDef::DMAmuxChannel`**
DMAMUX Channel Base Address
- **`DMAMUX_ChannelStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxChannelStatus`**
DMAMUX Channels Status Base Address
- **`uint32_t __DMA_HandleTypeDef::DMAmuxChannelStatusMask`**
DMAMUX Channel Status Mask
- **`DMAMUX_RequestGen_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGen`**
DMAMUX request generator Base Address
- **`DMAMUX_RequestGenStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGenStatus`**
DMAMUX request generator Status Address
- **`uint32_t __DMA_HandleTypeDef::DMAmuxRequestGenStatusMask`**
DMAMUX request generator Status mask

19.2 DMA Firmware driver API description

19.2.1

How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Stream (except for internal SRAM/FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests .
2. For a given Stream, program the required configuration through the following parameters: Transfer Direction, Source and Destination data formats, Circular, Normal or peripheral flow control mode, Stream Priority level, Source and Destination Increment mode, FIFO mode and its Threshold (if needed), Burst mode for Source and/or Destination (if needed) using HAL_DMA_Init() function.

Polling mode IO operation

- Use HAL_DMA_Start() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred
- Use HAL_DMA_PollForTransfer() to poll for the end of current transfer, in this case a fixed Timeout can be configured by User depending from his application.

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_DMA_Start_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_DMA_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of DMA handle structure).
- 1. Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- 2. Use HAL_DMA_Abort() function to abort the current transfer

Note:

In Memory-to-Memory transfer mode, Circular mode is not allowed.

Note:

The FIFO is used mainly to reduce bus usage and to allow data packing/unpacking: it is possible to set different Data Sizes for the Peripheral and the Memory (ie. you can set Half-Word data size for the peripheral to access its data register and set Word data size for the Memory to gain in access time. Each two half words will be packed and written in a single access to a Word in the Memory).

Note:

When FIFO is disabled, it is not allowed to configure different Data Sizes for Source and Destination. In this case the Peripheral Data Size will be applied to both Source and Destination.

DMA HAL driver macros list

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

- __HAL_DMA_ENABLE: Enable the specified DMA Stream.
- __HAL_DMA_DISABLE: Disable the specified DMA Stream.
- __HAL_DMA_GET_FS: Return the current DMA Stream FIFO filled level.
- __HAL_DMA_ENABLE_IT: Enable the specified DMA Stream interrupts.
- __HAL_DMA_DISABLE_IT: Disable the specified DMA Stream interrupts.
- __HAL_DMA_GET_IT_SOURCE: Check whether the specified DMA Stream interrupt has occurred or not.

Note:

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

19.2.2

Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Stream source and destination incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Stream priority value.

The HAL_DMA_Init() function follows the DMA configuration procedures as described in reference manual. The HAL_DMA_DelInit function allows to deinitialize the DMA stream.

This section contains the following APIs:

- [HAL_DMA_Init](#)
- [HAL_DMA_DelInit](#)

19.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Register and Unregister DMA callbacks
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request

This section contains the following APIs:

- [HAL_DMA_Start](#)
- [HAL_DMA_Start_IT](#)
- [HAL_DMA_Abort](#)
- [HAL_DMA_Abort_IT](#)
- [HAL_DMA_PollForTransfer](#)
- [HAL_DMA_IRQHandler](#)
- [HAL_DMA_RegisterCallback](#)
- [HAL_DMA_UnRegisterCallback](#)

19.2.4 State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code

This section contains the following APIs:

- [HAL_DMA_GetState](#)
- [HAL_DMA_GetError](#)

19.2.5 Detailed description of functions

HAL_DMA_Init

Function name

`HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)`

Function description

Initialize the DMA according to the specified parameters in the DMA_InitTypeDef and create the associated handle.

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMA_DeInit

Function name

`HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)`

Function description

DeInitializes the DMA peripheral.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMA_Start

Function name

`HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)`

Function description

Starts the DMA Transfer.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMA_Start_IT

Function name

`HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)`

Function description

Start the DMA Transfer with interrupt enabled.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMA_Abort

Function name

`HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)`

Function description

Aborts the DMA Transfer.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

Notes

- After disabling a DMA Stream, a check for wait until the DMA Stream is effectively disabled is added. If a Stream is disabled while a data transfer is ongoing, the current data will be transferred and the Stream will be effectively disabled only after the transfer of this single data is finished.

HAL_DMA_Abort_IT

Function name

`HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef * hdma)`

Function description

Aborts the DMA Transfer in Interrupt mode.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMA_PollForTransfer

Function name

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

Function description

Polling for transfer complete.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CompleteLevel:** Specifies the DMA level complete.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- The polling mode is kept in this version for legacy. it is recommended to use the IT model instead. This model could be used for debug purpose.

- The HAL_DMA_PollForTransfer API cannot be used in circular and double buffering mode (automatic circular mode).

HAL_DMA_IRQHandler

Function name

```
void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)
```

Function description

Handles DMA interrupt request.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **None:**

HAL_DMA_RegisterCallback

Function name

```
HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef * hdma,  
                                         HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef * _hdma) pCallback)
```

Function description

Register callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CallbackID:** User Callback identifier a DMA_HandleTypeDef structure as parameter.
- **pCallback:** pointer to private callback function which has pointer to a DMA_HandleTypeDef structure as parameter.

Return values

- **HAL:** status

HAL_DMA_UnRegisterCallback

Function name

```
HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef * hdma,  
                                             HAL_DMA_CallbackIDTypeDef CallbackID)
```

Function description

UnRegister callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CallbackID:** User Callback identifier a HAL_DMA_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status

HAL_DMA_GetState

Function name

`HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)`

Function description

Returns the DMA state.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** state

HAL_DMA_GetError

Function name

`uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)`

Function description

Return the DMA error code.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **DMA:** Error Code

19.3 DMA Firmware driver defines

19.3.1 DMA

DMA Data transfer direction

DMA_PERIPH_TO_MEMORY

Peripheral to memory direction

DMA_MEMORY_TO_PERIPH

Memory to peripheral direction

DMA_MEMORY_TO_MEMORY

Memory to memory direction

DMA Error Code

HAL_DMA_ERROR_NONE

No error

HAL_DMA_ERROR_TE

Transfer error

HAL_DMA_ERROR_FE

FIFO error

HAL_DMA_ERROR_DME

Direct Mode error

HAL_DMA_ERROR_TIMEOUT

Timeout error

HAL_DMA_ERROR_PARAM

Parameter error

HAL_DMA_ERROR_NO_XFER

Abort requested with no Xfer ongoing

HAL_DMA_ERROR_NOT_SUPPORTED

Not supported mode

HAL_DMA_ERROR_SYNC

DMAMUX sync overrun error

HAL_DMA_ERROR_REQGEN

DMAMUX request generator overrun error

HAL_DMA_ERROR_BUSY

DMA Busy error

DMA Exported Macros**_HAL_DMA_RESET_HANDLE_STATE****Description:**

- Reset DMA handle state.

Parameters:

- HANDLE: specifies the DMA handle.

Return value:

- None

_HAL_DMA_GET_FS**Description:**

- Return the current DMA Stream FIFO filled level.

Parameters:

- HANDLE: DMA handle

Return value:

- The: FIFO filling state.
 - DMA_FIFOStatus_Less1QuarterFull: when FIFO is less than 1 quarter-full and not empty.
 - DMA_FIFOStatus_1QuarterFull: if more than 1 quarter-full.
 - DMA_FIFOStatus_HalfFull: if more than 1 half-full.
 - DMA_FIFOStatus_3QuartersFull: if more than 3 quarters-full.
 - DMA_FIFOStatus_Empty: when FIFO is empty
 - DMA_FIFOStatus_Full: when FIFO is full

_HAL_DMA_ENABLE**Description:**

- Enable the specified DMA Stream.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- None

[__HAL_DMA_DISABLE](#)**Description:**

- Disable the specified DMA Stream.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- None

[__HAL_DMA_GET_TC_FLAG_INDEX](#)**Description:**

- Return the current DMA Stream transfer complete flag.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- The: specified transfer complete flag index.

[__HAL_DMA_GET_HT_FLAG_INDEX](#)**Description:**

- Return the current DMA Stream half transfer complete flag.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- The: specified half transfer complete flag index.

[__HAL_DMA_GET_TE_FLAG_INDEX](#)**Description:**

- Return the current DMA Stream transfer error flag.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- The: specified transfer error flag index.

[__HAL_DMA_GET_FE_FLAG_INDEX](#)**Description:**

- Return the current DMA Stream FIFO error flag.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- The: specified FIFO error flag index.

[__HAL_DMA_GET_DME_FLAG_INDEX](#)

Description:

- Return the current DMA Stream direct mode error flag.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- The: specified direct mode error flag index.

[__HAL_BDMA_GET_GI_FLAG_INDEX](#)

Description:

- Returns the current BDMA Channel Global interrupt flag.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- The: specified transfer error flag index.

[__HAL_DMA_GET_FLAG](#)

Description:

- Get the DMA Stream pending flags.

Parameters:

- `__HANDLE__`: DMA handle
- `__FLAG__`: Get the specified flag. This parameter can be any combination of the following values:
 - `DMA_FLAG_TCIFx`: Transfer complete flag.
 - `DMA_FLAG_HTIFx`: Half transfer complete flag.
 - `DMA_FLAG_TEIFx`: Transfer error flag.
 - `DMA_FLAG_DMEIFx`: Direct mode error flag.
 - `DMA_FLAG_FEIFx`: FIFO error flag. Where x can be `0_4`, `1_5`, `2_6` or `3_7` to select the DMA Stream flag.

Return value:

- The: state of FLAG (SET or RESET).

[__HAL_DMA_CLEAR_FLAG](#)

Description:

- Clear the DMA Stream pending flags.

Parameters:

- `__HANDLE__`: DMA handle
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
 - `DMA_FLAG_TCIFx`: Transfer complete flag.
 - `DMA_FLAG_HTIFx`: Half transfer complete flag.
 - `DMA_FLAG_TEIFx`: Transfer error flag.
 - `DMA_FLAG_DMEIFx`: Direct mode error flag.
 - `DMA_FLAG_FEIFx`: FIFO error flag. Where x can be `0_4`, `1_5`, `2_6` or `3_7` to select the DMA Stream flag.

Return value:

- None

D2_TO_D3_DMA_IT

`_HAL_DMA_D3_ENABLE_IT`

`_HAL_DMA_D2_ENABLE_IT`

`_HAL_DMA_ENABLE_IT`

Description:

- Enable the specified DMA Stream interrupts.

Parameters:

- `_HANDLE_`: DMA handle
- `_INTERRUPT_`: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be one of the following values:
 - `DMA_IT_TC`: Transfer complete interrupt mask.
 - `DMA_IT_HT`: Half transfer complete interrupt mask.
 - `DMA_IT_TE`: Transfer error interrupt mask.
 - `DMA_IT_FE`: FIFO error interrupt mask.
 - `DMA_IT_DME`: Direct mode error interrupt.

Return value:

- None

`_HAL_DMA_D3_DISABLE_IT`

`_HAL_DMA_D2_DISABLE_IT`

`_HAL_DMA_DISABLE_IT`

Description:

- Disable the specified DMA Stream interrupts.

Parameters:

- `_HANDLE_`: DMA handle
- `_INTERRUPT_`: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be one of the following values:
 - `DMA_IT_TC`: Transfer complete interrupt mask.
 - `DMA_IT_HT`: Half transfer complete interrupt mask.
 - `DMA_IT_TE`: Transfer error interrupt mask.
 - `DMA_IT_FE`: FIFO error interrupt mask.
 - `DMA_IT_DME`: Direct mode error interrupt.

Return value:

- None

`_HAL_DMA_D3_GET_IT_SOURCE`

`_HAL_DMA_D2_GET_IT_SOURCE`

`_HAL_DMA_GET_IT_SOURCE`

Description:

- Check whether the specified DMA Stream interrupt is enabled or not.

Parameters:

- `_HANDLE_`: DMA handle

- `__INTERRUPT__`: specifies the DMA interrupt source to check. This parameter can be one of the following values:
 - DMA_IT_TC: Transfer complete interrupt mask.
 - DMA_IT_HT: Half transfer complete interrupt mask.
 - DMA_IT_TE: Transfer error interrupt mask.
 - DMA_IT_FE: FIFO error interrupt mask.
 - DMA_IT_DME: Direct mode error interrupt.

Return value:

- The: state of DMA_IT.

[__HAL_DMA_SET_COUNTER](#)

Description:

- Writes the number of data units to be transferred on the DMA Stream.

Parameters:

- `__HANDLE__`: DMA handle
- `__COUNTER__`: Number of data units to be transferred (from 0 to 65535) Number of data items depends only on the Peripheral data format.

Return value:

- The: number of remaining data units in the current DMAy Streamx transfer.

Notes:

- If Peripheral data format is Bytes: number of data units is equal to total number of bytes to be transferred.
If Peripheral data format is Half-Word: number of data units is equal to total number of bytes to be transferred / 2.
If Peripheral data format is Word: number of data units is equal to total number of bytes to be transferred / 4.

[__HAL_DMA_GET_COUNTER](#)

Description:

- Returns the number of remaining data units in the current DMAy Streamx transfer.

Parameters:

- `__HANDLE__`: DMA handle

Return value:

- The: number of remaining data units in the current DMA Stream transfer.

DMA FIFO direct mode

[DMA_FIFOMODE_DISABLE](#)

FIFO mode disable

[DMA_FIFOMODE_ENABLE](#)

FIFO mode enable

DMA FIFO threshold level

[DMA_FIFO_THRESHOLD_1QUARTERFULL](#)

FIFO threshold 1 quart full configuration

[DMA_FIFO_THRESHOLD_HALFFULL](#)

FIFO threshold half full configuration

[DMA_FIFO_THRESHOLD_3QUARTERSFULL](#)

FIFO threshold 3 quarts full configuration

DMA_FIFO_THRESHOLD_FULL

FIFO threshold full configuration

DMA flag definitions

DMA_FLAG_FEIF0_4

DMA_FLAG_DMEIF0_4

DMA_FLAG_TEIF0_4

DMA_FLAG_HTIF0_4

DMA_FLAG_TCIF0_4

DMA_FLAG_FEIF1_5

DMA_FLAG_DMEIF1_5

DMA_FLAG_TEIF1_5

DMA_FLAG_HTIF1_5

DMA_FLAG_TCIF1_5

DMA_FLAG_FEIF2_6

DMA_FLAG_DMEIF2_6

DMA_FLAG_TEIF2_6

DMA_FLAG_HTIF2_6

DMA_FLAG_TCIF2_6

DMA_FLAG_FEIF3_7

DMA_FLAG_DMEIF3_7

DMA_FLAG_TEIF3_7

DMA_FLAG_HTIF3_7

DMA_FLAG_TCIF3_7

DMA interrupt enable definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA_IT_DME

DMA_IT_FE

*DMA Memory burst***DMA_MBURST_SINGLE****DMA_MBURST_INC4****DMA_MBURST_INC8****DMA_MBURST_INC16***DMA Memory data size***DMA_MDATAALIGN_BYTE**

Memory data alignment: Byte

DMA_MDATAALIGN_HALFWORD

Memory data alignment: HalfWord

DMA_MDATAALIGN_WORD

Memory data alignment: Word

*DMA Memory incremented mode***DMA_MINC_ENABLE**

Memory increment mode enable

DMA_MINC_DISABLE

Memory increment mode disable

*DMA mode***DMA_NORMAL**

Normal mode

DMA_CIRCULAR

Circular mode

DMA_PFCTRL

Peripheral flow control mode

*DMA Peripheral burst***DMA_PBURST_SINGLE****DMA_PBURST_INC4****DMA_PBURST_INC8****DMA_PBURST_INC16***DMA Peripheral data size***DMA_PDATAALIGN_BYTE**

Peripheral data alignment: Byte

DMA_PDATAALIGN_HALFWORD

Peripheral data alignment: HalfWord

DMA_PDATAALIGN_WORD

Peripheral data alignment: Word

DMA Peripheral incremented mode**DMA_PINC_ENABLE**

Peripheral increment mode enable

DMA_PINC_DISABLE

Peripheral increment mode disable

DMA Priority level**DMA_PRIORITY_LOW**

Priority level: Low

DMA_PRIORITY_MEDIUM

Priority level: Medium

DMA_PRIORITY_HIGH

Priority level: High

DMA_PRIORITY VERY HIGH

Priority level: Very High

DMA Request selection**DMA_REQUEST_MEM2MEM**

memory to memory transfer

DMA_REQUEST_GENERATOR0

DMAMUX1 request generator 0

DMA_REQUEST_GENERATOR1

DMAMUX1 request generator 1

DMA_REQUEST_GENERATOR2

DMAMUX1 request generator 2

DMA_REQUEST_GENERATOR3

DMAMUX1 request generator 3

DMA_REQUEST_GENERATOR4

DMAMUX1 request generator 4

DMA_REQUEST_GENERATOR5

DMAMUX1 request generator 5

DMA_REQUEST_GENERATOR6

DMAMUX1 request generator 6

DMA_REQUEST_GENERATOR7

DMAMUX1 request generator 7

DMA_REQUEST_ADC1

DMAMUX1 ADC1 request

DMA_REQUEST_ADC2

DMAMUX1 ADC2 request

DMA_REQUEST_TIM1_CH1

DMAMUX1 TIM1 CH1 request

DMA_REQUEST_TIM1_CH2

DMAMUX1 TIM1 CH2 request

DMA_REQUEST_TIM1_CH3

DMAMUX1 TIM1 CH3 request

DMA_REQUEST_TIM1_CH4

DMAMUX1 TIM1 CH4 request

DMA_REQUEST_TIM1_UP

DMAMUX1 TIM1 UP request

DMA_REQUEST_TIM1_TRIG

DMAMUX1 TIM1 TRIG request

DMA_REQUEST_TIM1_COM

DMAMUX1 TIM1 COM request

DMA_REQUEST_TIM2_CH1

DMAMUX1 TIM2 CH1 request

DMA_REQUEST_TIM2_CH2

DMAMUX1 TIM2 CH2 request

DMA_REQUEST_TIM2_CH3

DMAMUX1 TIM2 CH3 request

DMA_REQUEST_TIM2_CH4

DMAMUX1 TIM2 CH4 request

DMA_REQUEST_TIM2_UP

DMAMUX1 TIM2 UP request

DMA_REQUEST_TIM3_CH1

DMAMUX1 TIM3 CH1 request

DMA_REQUEST_TIM3_CH2

DMAMUX1 TIM3 CH2 request

DMA_REQUEST_TIM3_CH3

DMAMUX1 TIM3 CH3 request

DMA_REQUEST_TIM3_CH4

DMAMUX1 TIM3 CH4 request

DMA_REQUEST_TIM3_UP

DMAMUX1 TIM3 UP request

DMA_REQUEST_TIM3_TRIG

DMAMUX1 TIM3 TRIG request

DMA_REQUEST_TIM4_CH1

DMAMUX1 TIM4 CH1 request

DMA_REQUEST_TIM4_CH2

DMAMUX1 TIM4 CH2 request

DMA_REQUEST_TIM4_CH3

DMAMUX1 TIM4 CH3 request

DMA_REQUEST_TIM4_UP

DMAMUX1 TIM4 UP request

DMA_REQUEST_I2C1_RX

DMAMUX1 I2C1 RX request

DMA_REQUEST_I2C1_TX

DMAMUX1 I2C1 TX request

DMA_REQUEST_I2C2_RX

DMAMUX1 I2C2 RX request

DMA_REQUEST_I2C2_TX

DMAMUX1 I2C2 TX request

DMA_REQUEST_SPI1_RX

DMAMUX1 SPI1 RX request

DMA_REQUEST_SPI1_TX

DMAMUX1 SPI1 TX request

DMA_REQUEST_SPI2_RX

DMAMUX1 SPI2 RX request

DMA_REQUEST_SPI2_TX

DMAMUX1 SPI2 TX request

DMA_REQUEST_USART1_RX

DMAMUX1 USART1 RX request

DMA_REQUEST_USART1_TX

DMAMUX1 USART1 TX request

DMA_REQUEST_USART2_RX

DMAMUX1 USART2 RX request

DMA_REQUEST_USART2_TX

DMAMUX1 USART2 TX request

DMA_REQUEST_USART3_RX

DMAMUX1 USART3 RX request

DMA_REQUEST_USART3_TX

DMAMUX1 USART3 TX request

DMA_REQUEST_TIM8_CH1

DMAMUX1 TIM8 CH1 request

DMA_REQUEST_TIM8_CH2

DMAMUX1 TIM8 CH2 request

DMA_REQUEST_TIM8_CH3

DMAMUX1 TIM8 CH3 request

DMA_REQUEST_TIM8_CH4

DMAMUX1 TIM8 CH4 request

DMA_REQUEST_TIM8_UP

DMAMUX1 TIM8 UP request

DMA_REQUEST_TIM8_TRIG

DMAMUX1 TIM8 TRIG request

DMA_REQUEST_TIM8_COM

DMAMUX1 TIM8 COM request

DMA_REQUEST_TIM5_CH1

DMAMUX1 TIM5 CH1 request

DMA_REQUEST_TIM5_CH2

DMAMUX1 TIM5 CH2 request

DMA_REQUEST_TIM5_CH3

DMAMUX1 TIM5 CH3 request

DMA_REQUEST_TIM5_CH4

DMAMUX1 TIM5 CH4 request

DMA_REQUEST_TIM5_UP

DMAMUX1 TIM5 UP request

DMA_REQUEST_TIM5_TRIG

DMAMUX1 TIM5 TRIG request

DMA_REQUEST_SPI3_RX

DMAMUX1 SPI3 RX request

DMA_REQUEST_SPI3_TX

DMAMUX1 SPI3 TX request

DMA_REQUEST_UART4_RX

DMAMUX1 UART4 RX request

DMA_REQUEST_UART4_TX

DMAMUX1 UART4 TX request

DMA_REQUEST_UART5_RX

DMAMUX1 UART5 RX request

DMA_REQUEST_UART5_TX

DMAMUX1 UART5 TX request

DMA_REQUEST_DAC1_CH1

DMAMUX1 DAC1 Channel 1 request

DMA_REQUEST_DAC1_CH2

DMAMUX1 DAC1 Channel 2 request

DMA_REQUEST_TIM6_UP

DMAMUX1 TIM6 UP request

DMA_REQUEST_TIM7_UP

DMAMUX1 TIM7 UP request

DMA_REQUEST_USART6_RX

DMAMUX1 USART6 RX request

DMA_REQUEST_USART6_TX

DMAMUX1 USART6 TX request

DMA_REQUEST_I2C3_RX

DMAMUX1 I2C3 RX request

DMA_REQUEST_I2C3_TX

DMAMUX1 I2C3 TX request

DMA_REQUEST_DCMI

DMAMUX1 DCMI request

DMA_REQUEST_CRYP_IN

DMAMUX1 CRYP IN request

DMA_REQUEST_CRYP_OUT

DMAMUX1 CRYP OUT request

DMA_REQUEST_HASH_IN

DMAMUX1 HASH IN request

DMA_REQUEST_UART7_RX

DMAMUX1 UART7 RX request

DMA_REQUEST_UART7_TX

DMAMUX1 UART7 TX request

DMA_REQUEST_UART8_RX

DMAMUX1 UART8 RX request

DMA_REQUEST_UART8_TX

DMAMUX1 UART8 TX request

DMA_REQUEST_SPI4_RX

DMAMUX1 SPI4 RX request

DMA_REQUEST_SPI4_TX

DMAMUX1 SPI4 TX request

DMA_REQUEST_SPI5_RX

DMAMUX1 SPI5 RX request

DMA_REQUEST_SPI5_TX

DMAMUX1 SPI5 TX request

DMA_REQUEST_SAI1_A

DMAMUX1 SAI1 A request

DMA_REQUEST_SAI1_B

DMAMUX1 SAI1 B request

DMA_REQUEST_SAI2_A

DMAMUX1 SAI2 A request

DMA_REQUEST_SAI2_B

DMAMUX1 SAI2 B request

DMA_REQUEST_SWPMI_RX

DMAMUX1 SWPMI RX request

DMA_REQUEST_SWPMI_TX

DMAMUX1 SWPMI TX request

DMA_REQUEST_SPDIF_RX_DT

DMAMUX1 SPDIF RXDT request

DMA_REQUEST_SPDIF_RX_CS

DMAMUX1 SPDIF RXCS request

DMA_REQUEST_HRTIM_MASTER

DMAMUX1 HRTIM1 Master request 1

DMA_REQUEST_HRTIM_TIMER_A

DMAMUX1 HRTIM1 TimerA request 2

DMA_REQUEST_HRTIM_TIMER_B

DMAMUX1 HRTIM1 TimerB request 3

DMA_REQUEST_HRTIM_TIMER_C

DMAMUX1 HRTIM1 TimerC request 4

DMA_REQUEST_HRTIM_TIMER_D

DMAMUX1 HRTIM1 TimerD request 5

DMA_REQUEST_HRTIM_TIMER_E

DMAMUX1 HRTIM1 TimerE request 6

DMA_REQUEST_DFSDM1_FLT0

DMAMUX1 DFSDM Filter0 request

DMA_REQUEST_DFSDM1_FLT1

DMAMUX1 DFSDM Filter1 request

DMA_REQUEST_DFSDM1_FLT2

DMAMUX1 DFSDM Filter2 request

DMA_REQUEST_DFSDM1_FLT3

DMAMUX1 DFSDM Filter3 request

DMA_REQUEST_TIM15_CH1

DMAMUX1 TIM15 CH1 request

DMA_REQUEST_TIM15_UP

DMAMUX1 TIM15 UP request

DMA_REQUEST_TIM15_TRIG

DMAMUX1 TIM15 TRIG request

DMA_REQUEST_TIM15_COM

DMAMUX1 TIM15 COM request

DMA_REQUEST_TIM16_CH1

DMAMUX1 TIM16 CH1 request

DMA_REQUEST_TIM16_UP

DMAMUX1 TIM16 UP request

DMA_REQUEST_TIM17_CH1

DMAMUX1 TIM17 CH1 request

DMA_REQUEST_TIM17_UP

DMAMUX1 TIM17 UP request

DMA_REQUEST_SAI3_A

DMAMUX1 SAI3 A request

DMA_REQUEST_SAI3_B

DMAMUX1 SAI3 B request

DMA_REQUEST_ADC3

DMAMUX1 ADC3 request

BDMA_REQUEST_MEM2MEM

memory to memory transfer

BDMA_REQUEST_GENERATOR0

DMAMUX2 request generator 0

BDMA_REQUEST_GENERATOR1

DMAMUX2 request generator 1

BDMA_REQUEST_GENERATOR2

DMAMUX2 request generator 2

BDMA_REQUEST_GENERATOR3

DMAMUX2 request generator 3

BDMA_REQUEST_GENERATOR4

DMAMUX2 request generator 4

BDMA_REQUEST_GENERATOR5

DMAMUX2 request generator 5

BDMA_REQUEST_GENERATOR6

DMAMUX2 request generator 6

BDMA_REQUEST_GENERATOR7

DMAMUX2 request generator 7

BDMA_REQUEST_LPUART1_RX

DMAMUX2 LP_UART1_RX request

BDMA_REQUEST_LPUART1_TX

DMAMUX2 LP_UART1_TX request

BDMA_REQUEST_SPI6_RX

DMAMUX2 SPI6 RX request

BDMA_REQUEST_SPI6_TX

DMAMUX2 SPI6 TX request

BDMA_REQUEST_I2C4_RX

DMAMUX2 I2C4 RX request

BDMA_REQUEST_I2C4_TX

DMAMUX2 I2C4 TX request

BDMA_REQUEST_SAI4_A

DMAMUX2 SAI4 A request

BDMA_REQUEST_SAI4_B

DMAMUX2 SAI4 B request

BDMA_REQUEST_ADC3

DMAMUX2 ADC3 request

20 HAL DMA Extension Driver

20.1 DMAEx Firmware driver registers structures

20.1.1 HAL_DMA_MuxSyncConfigTypeDef

Data Fields

- *uint32_t SyncSignalID*
- *uint32_t SyncPolarity*
- *FunctionalState SyncEnable*
- *FunctionalState EventEnable*
- *uint32_t RequestNumber*

Field Documentation

- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncSignalID***

Specifies the synchronization signal gating the DMA request in periodic mode. This parameter can be a value of **DMAEx MUX SyncSignalID selection**

- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncPolarity***

Specifies the polarity of the signal on which the DMA request is synchronized. This parameter can be a value of **DMAEx MUX SyncPolarity selection**

- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::SyncEnable***

Specifies if the synchronization shall be enabled or disabled. This parameter can take the value ENABLE or DISABLE

- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::EventEnable***

Specifies if an event shall be generated once the RequestNumber is reached. This parameter can take the value ENABLE or DISABLE

- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::RequestNumber***

Specifies the number of DMA request that will be authorized after a sync event. This parameters can be in the range 1 to 32

20.1.2 HAL_DMA_MuxRequestGeneratorConfigTypeDef

Data Fields

- *uint32_t SignalID*
- *uint32_t Polarity*
- *uint32_t RequestNumber*

Field Documentation

- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::SignalID***

Specifies the ID of the signal used for DMAMUX request generator. This parameter can be a value of **DMAEx MUX SignalGeneratorID selection**

- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::Polarity***

Specifies the polarity of the signal on which the request is generated. This parameter can be a value of **DMAEx MUX RequestGeneneratorPolarity selection**

- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::RequestNumber***

Specifies the number of DMA request that will be generated after a signal event. This parameters can be in the range 1 to 32

20.2 DMAEx Firmware driver API description

20.2.1 How to use this driver

The DMA Extension HAL driver can be used as follows:

- Start a multi buffer transfer using the HAL_DMA_MultiBufferStart() function for polling mode or HAL_DMA_MultiBufferStart_IT() for interrupt mode.
- Configure the DMA_MUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMA_MUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function. Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.
- To handle the DMAMUX Interrupts, the function HAL_DMAEx_MUX_IRQHandler should be called from the DMAMUX IRQ handler i.e DMAMUX1_OVR_IRQHandler or DMAMUX2_OVR_IRQHandler . As only one interrupt line is available for all DMAMUX channels and request generators , HAL_DMA_MUX_IRQHandler should be called with, as parameter, the appropriate DMA handle as many as used DMAs in the user project (exception done if a given DMA is not using the DMAMUX SYNC block neither a request generator)

Note: *In Memory-to-Memory transfer mode, Multi (Double) Buffer mode is not allowed.*

Note: *When Multi (Double) Buffer mode is enabled, the transfer is circular by default.*

Note: *In Multi (Double) buffer mode, it is possible to update the base address for the AHB memory port on the fly (DMA_SxM0AR or DMA_SxM1AR) when the stream is enabled.*

Note: *Multi (Double) buffer mode is only possible with D2 DMAs i.e DMA1 or DMA2. not BDMA. Multi (Double) buffer mode is not possible with D3 BDMA.*

20.2.2 Extended features functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start MultiBuffer DMA transfer
- Configure the source, destination address and data length and Start MultiBuffer DMA transfer with interrupt
- Change on the fly the memory0 or memory1 address.
- Configure the DMA_MUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMA_MUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function.
- Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.
- Handle DMAMUX interrupts using HAL_DMAEx_MUX_IRQHandler : should be called from the DMAMUX IRQ handler i.e DMAMUX1_OVR_IRQHandler or DMAMUX2_OVR_IRQHandler

This section contains the following APIs:

- [**HAL_DMAEx_MultiBufferStart**](#)
- [**HAL_DMAEx_MultiBufferStart_IT**](#)
- [**HAL_DMAEx_ChangeMemory**](#)
- [**HAL_DMAEx_ConfigMuxSync**](#)
- [**HAL_DMAEx_ConfigMuxRequestGenerator**](#)
- [**HAL_DMAEx_EnableMuxRequestGenerator**](#)
- [**HAL_DMAEx_DisableMuxRequestGenerator**](#)
- [**HAL_DMAEx_MUX_IRQHandler**](#)

20.2.3 Detailed description of functions

HAL_DMAEx_MultiBufferStart

Function name

HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)

Function description

Starts the multi_buffer DMA Transfer.

Parameters

- **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **SecondMemAddress:** The second memory Buffer address in case of multi buffer Transfer
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMAEx_MultiBufferStart_IT

Function name

HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)

Function description

Starts the multi_buffer DMA Transfer with interrupt enabled.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **SecondMemAddress:** The second memory Buffer address in case of multi buffer Transfer
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMAEx_ChangeMemory

Function name

HAL_StatusTypeDef HAL_DMAEx_ChangeMemory (DMA_HandleTypeDef * hdma, uint32_t Address, HAL_DMA_MemoryTypeDef memory)

Function description

Change the memory0 or memory1 address on the fly.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **Address:** The new address

- **memory:** the memory to be changed, This parameter can be one of the following values: MEMORY0 / MEMORY1

Return values

- **HAL:** status

Notes

- The MEMORY0 address can be changed only when the current transfer use MEMORY1 and the MEMORY1 address can be changed only when the current transfer use MEMORY0.

HAL_DMAEx_ConfigMuxSync

Function name

**HAL_StatusTypeDef HAL_DMAEx_ConfigMuxSync (DMA_HandleTypeDef * hdma,
HAL_DMA_MuxSyncConfigTypeDef * pSyncConfig)**

Function description

Configure the DMAMUX synchronization parameters for a given DMA stream (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **pSyncConfig:** : pointer to HAL_DMA_MuxSyncConfigTypeDef : contains the DMAMUX synchronization parameters

Return values

- **HAL:** status

HAL_DMAEx_ConfigMuxRequestGenerator

Function name

**HAL_StatusTypeDef HAL_DMAEx_ConfigMuxRequestGenerator (DMA_HandleTypeDef * hdma,
HAL_DMA_MuxRequestGeneratorConfigTypeDef * pRequestGeneratorConfig)**

Function description

Configure the DMAMUX request generator block used by the given DMA stream (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **pRequestGeneratorConfig:** : pointer to HAL_DMA_MuxRequestGeneratorConfigTypeDef : contains the request generator parameters.

Return values

- **HAL:** status

HAL_DMAEx_EnableMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_EnableMuxRequestGenerator (DMA_HandleTypeDef * hdma)

Function description

Enable the DMAMUX request generator block used by the given DMA stream (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMAEx_DisableMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_DisableMuxRequestGenerator (DMA_HandleTypeDef * hdma)

Function description

Disable the DMAMUX request generator block used by the given DMA stream (instance).

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMAEx_MUX_IRQHandler

Function name

void HAL_DMAEx_MUX_IRQHandler (DMA_HandleTypeDef * hdma)

Function description

Handles DMAMUX interrupt request.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **None:**

20.3 DMAEx Firmware driver defines

20.3.1 DMAEx

DMAEx MUX RequestGeneneratorPolarity selection

HAL_DMAMUX_REQ_GEN_NO_EVENT

block request generator events

HAL_DMAMUX_REQ_GEN_RISING

generate request on rising edge events

HAL_DMAMUX_REQ_GEN_FALLING

generate request on falling edge events

HAL_DMAMUX_REQ_GEN_RISING_FALLING

generate request on rising and falling edge events

DMAEx MUX SignalGeneratorID selection

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH0_EVT

D2 domain Request generator Signal is DMAMUX1 Channel0 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH1_EVT

D2 domain Request generator Signal is DMAMUX1 Channel1 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH2_EVT

D2 domain Request generator Signal is DMAMUX1 Channel2 Event

HAL_DMAMUX1_REQ_GEN_LPTIM1_OUT

D2 domain Request generator Signal is LPTIM1 OUT

HAL_DMAMUX1_REQ_GEN_LPTIM2_OUT

D2 domain Request generator Signal is LPTIM2 OUT

HAL_DMAMUX1_REQ_GEN_LPTIM3_OUT

D2 domain Request generator Signal is LPTIM3 OUT

HAL_DMAMUX1_REQ_GEN_EXTI0

D2 domain Request generator Signal is EXTI0 IT

HAL_DMAMUX1_REQ_GEN_TIM12_TRGO

D2 domain Request generator Signal is TIM12 TRGO

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH0_EVT

D3 domain Request generator Signal is DMAMUX2 Channel0 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH1_EVT

D3 domain Request generator Signal is DMAMUX2 Channel1 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH2_EVT

D3 domain Request generator Signal is DMAMUX2 Channel2 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH3_EVT

D3 domain Request generator Signal is DMAMUX2 Channel3 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH4_EVT

D3 domain Request generator Signal is DMAMUX2 Channel4 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH5_EVT

D3 domain Request generator Signal is DMAMUX2 Channel5 Event

HAL_DMAMUX2_REQ_GEN_DMAMUX2_CH6_EVT

D3 domain Request generator Signal is DMAMUX2 Channel6 Event

HAL_DMAMUX2_REQ_GEN_LPUART1_RX_WKUP

D3 domain Request generator Signal is LPUART1 RX Wakeup

HAL_DMAMUX2_REQ_GEN_LPUART1_TX_WKUP

D3 domain Request generator Signal is LPUART1 TX Wakeup

HAL_DMAMUX2_REQ_GEN_LPTIM2_WKUP

D3 domain Request generator Signal is LPTIM2 Wakeup

HAL_DMAMUX2_REQ_GEN_LPTIM2_OUT

D3 domain Request generator Signal is LPTIM2 OUT

HAL_DMAMUX2_REQ_GEN_LPTIM3_WKUP

D3 domain Request generator Signal is LPTIM3 Wakeup

HAL_DMAMUX2_REQ_GEN_LPTIM3_OUT

D3 domain Request generator Signal is LPTIM3 OUT

HAL_DMAMUX2_REQ_GEN_LPTIM4_WKUP

D3 domain Request generator Signal is LPTIM4 Wakeup

HAL_DMAMUX2_REQ_GEN_LPTIM5_WKUP

D3 domain Request generator Signal is LPTIM5 Wakeup

HAL_DMAMUX2_REQ_GEN_I2C4_WKUP

D3 domain Request generator Signal is I2C4 Wakeup

HAL_DMAMUX2_REQ_GEN_SPI6_WKUP

D3 domain Request generator Signal is SPI6 Wakeup

HAL_DMAMUX2_REQ_GEN_COMP1_OUT

D3 domain Request generator Signal is Comparator 1 output

HAL_DMAMUX2_REQ_GEN_COMP2_OUT

D3 domain Request generator Signal is Comparator 2 output

HAL_DMAMUX2_REQ_GEN_RTC_WKUP

D3 domain Request generator Signal is RTC Wakeup

HAL_DMAMUX2_REQ_GEN_EXTI0

D3 domain Request generator Signal is EXTI0

HAL_DMAMUX2_REQ_GEN_EXTI2

D3 domain Request generator Signal is EXTI2

HAL_DMAMUX2_REQ_GEN_I2C4_IT_EVT

D3 domain Request generator Signal is I2C4 IT Event

HAL_DMAMUX2_REQ_GEN_SPI6_IT

D3 domain Request generator Signal is SPI6 IT

HAL_DMAMUX2_REQ_GEN_LPUART1_TX_IT

D3 domain Request generator Signal is LPUART1 Tx IT

HAL_DMAMUX2_REQ_GEN_LPUART1_RX_IT

D3 domain Request generator Signal is LPUART1 Rx IT

HAL_DMAMUX2_REQ_GEN_ADC3_IT

D3 domain Request generator Signal is ADC3 IT

HAL_DMAMUX2_REQ_GEN_ADC3_AWD1_OUT

D3 domain Request generator Signal is ADC3 Analog Watchdog 1 output

HAL_DMAMUX2_REQ_GEN_BDMA_CH0_IT

D3 domain Request generator Signal is BDMA Channel 0 IT

HAL_DMAMUX2_REQ_GEN_BDMA_CH1_IT

D3 domain Request generator Signal is BDMA Channel 1 IT

DMAEx MUX SyncPolarity selection

HAL_DMAMUX_SYNC_NO_EVENT

block synchronization events

HAL_DMAMUX_SYNC_RISING

synchronize with rising edge events

HAL_DMAMUX_SYNC_FALLING

synchronize with falling edge events

HAL_DMAMUX_SYNC_RISING_FALLING

synchronize with rising and falling edge events

DMAEx MUX SyncSignalID selection**HAL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT**

D2 Domain synchronization Signal is DMAMUX1 Channel0 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT

D2 Domain synchronization Signal is DMAMUX1 Channel1 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT

D2 Domain synchronization Signal is DMAMUX1 Channel2 Event

HAL_DMAMUX1_SYNC_LPTIM1_OUT

D2 Domain synchronization Signal is LPTIM1 OUT

HAL_DMAMUX1_SYNC_LPTIM2_OUT

D2 Domain synchronization Signal is LPTIM2 OUT

HAL_DMAMUX1_SYNC_LPTIM3_OUT

D2 Domain synchronization Signal is LPTIM3 OUT

HAL_DMAMUX1_SYNC_EXTIO

D2 Domain synchronization Signal is EXTIO IT

HAL_DMAMUX1_SYNC_TIM12_TRGO

D2 Domain synchronization Signal is TIM12 TRGO

HAL_DMAMUX2_SYNC_DMAMUX2_CH0_EVT

D3 Domain synchronization Signal is DMAMUX2 Channel0 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH1_EVT

D3 Domain synchronization Signal is DMAMUX2 Channel1 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH2_EVT

D3 Domain synchronization Signal is DMAMUX2 Channel2 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH3_EVT

D3 Domain synchronization Signal is DMAMUX2 Channel3 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH4_EVT

D3 Domain synchronization Signal is DMAMUX2 Channel4 Event

HAL_DMAMUX2_SYNC_DMAMUX2_CH5_EVT

D3 Domain synchronization Signal is DMAMUX2 Channel5 Event

HAL_DMAMUX2_SYNC_LPUART1_RX_WKUP

D3 Domain synchronization Signal is LPUART1 RX Wakeup

HAL_DMAMUX2_SYNC_LPUART1_TX_WKUP

D3 Domain synchronization Signal is LPUART1 TX Wakeup

HAL_DMAMUX2_SYNC_LPTIM2_OUT

D3 Domain synchronization Signal is LPTIM2 output

HAL_DMAMUX2_SYNC_LPTIM3_OUT

D3 Domain synchronization Signal is LPTIM3 output

HAL_DMAMUX2_SYNC_I2C4_WKUP

D3 Domain synchronization Signal is I2C4 Wakeup

HAL_DMAMUX2_SYNC_SPI6_WKUP

D3 Domain synchronization Signal is SPI6 Wakeup

HAL_DMAMUX2_SYNC_COMP1_OUT

D3 Domain synchronization Signal is Comparator 1 output

HAL_DMAMUX2_SYNC_RTC_WKUP

D3 Domain synchronization Signal is RTC Wakeup

HAL_DMAMUX2_SYNC_EXTI0

D3 Domain synchronization Signal is EXTI0 IT

HAL_DMAMUX2_SYNC_EXTI2

D3 Domain synchronization Signal is EXTI2 IT

21 HAL ETH Generic Driver

21.1 ETH Firmware driver registers structures

21.1.1 __ETH_BufferTypeDef

Data Fields

- `uint8_t * buffer`
- `uint32_t len`
- `struct __ETH_BufferTypeDef * next`

Field Documentation

- `uint8_t* __ETH_BufferTypeDef::buffer`
- `uint32_t __ETH_BufferTypeDef::len`
- `struct __ETH_BufferTypeDef* __ETH_BufferTypeDef::next`

21.1.2 ETH_TxDescListTypeDef

Data Fields

- `uint32_t TxDesc`
- `uint32_t CurTxDesc`

Field Documentation

- `uint32_t ETH_TxDescListTypeDef::TxDesc[ETH_TX_DESC_CNT]`
- `uint32_t ETH_TxDescListTypeDef::CurTxDesc`

21.1.3 ETH_TxPacketConfig

Data Fields

- `uint32_t Attributes`
- `uint32_t Length`
- `ETH_BufferTypeDef * TxBuffer`
- `uint32_t SrcAddrCtrl`
- `uint32_t CRCPadCtrl`
- `uint32_t ChecksumCtrl`
- `uint32_t MaxSegmentSize`
- `uint32_t PayloadLen`
- `uint32_t TCPHeaderLen`
- `uint32_t VlanTag`
- `uint32_t VlanCtrl`
- `uint32_t InnerVlanTag`
- `uint32_t InnerVlanCtrl`

Field Documentation

- `uint32_t ETH_TxPacketConfig::Attributes`

Tx packet HW features capabilities. This parameter can be a combination of **ETH Tx Packet Attributes**

- **`uint32_t ETH_TxPacketConfig::Length`**
Total packet length
- **`ETH_BufferTypeDef* ETH_TxPacketConfig::TxBuffer`**
Tx buffers pointers
- **`uint32_t ETH_TxPacketConfig::SrcAddrCtrl`**
Specifies the source address insertion control. This parameter can be a value of **ETH Tx Packet Source Addr Control**
- **`uint32_t ETH_TxPacketConfig::CRCPadCtrl`**
Specifies the CRC and Pad insertion and replacement control. This parameter can be a value of **ETH Tx Packet CRC Pad Control**
- **`uint32_t ETH_TxPacketConfig::ChecksumCtrl`**
Specifies the checksum insertion control. This parameter can be a value of **ETH Tx Packet Checksum Control**
- **`uint32_t ETH_TxPacketConfig::MaxSegmentSize`**
Sets TCP maximum segment size only when TCP segmentation is enabled. This parameter can be a value from 0x0 to 0x3FFF
- **`uint32_t ETH_TxPacketConfig::PayloadLen`**
Sets Total payload length only when TCP segmentation is enabled. This parameter can be a value from 0x0 to 0xFFFF
- **`uint32_t ETH_TxPacketConfig::TCPHeaderLen`**
Sets TCP header length only when TCP segmentation is enabled. This parameter can be a value from 0x5 to 0xF
- **`uint32_t ETH_TxPacketConfig::VlanTag`**
Sets VLAN Tag only when VLAN is enabled. This parameter can be a value from 0x0 to 0xFFFF
- **`uint32_t ETH_TxPacketConfig::VlanCtrl`**
Specifies VLAN Tag insertion control only when VLAN is enabled. This parameter can be a value of **ETH Tx Packet VLAN Control**
- **`uint32_t ETH_TxPacketConfig::InnerVlanTag`**
Sets Inner VLAN Tag only when Inner VLAN is enabled. This parameter can be a value from 0x0 to 0xFFFF
- **`uint32_t ETH_TxPacketConfig::InnerVlanCtrl`**
Specifies Inner VLAN Tag insertion control only when Inner VLAN is enabled. This parameter can be a value of **ETH Tx Packet Inner VLAN Control**

21.1.4 ETH_RxDescListTypeDef

Data Fields

- **`uint32_t RxDesc`**
- **`uint32_t CurRxDesc`**
- **`uint32_t FirstAppDesc`**
- **`uint32_t AppDescNbr`**
- **`uint32_t ApplicationContextDesc`**
- **`uint32_t ItMode`**

Field Documentation

- **`uint32_t ETH_RxDescListTypeDef::RxDesc[ETH_RX_DESC_CNT]`**

- `uint32_t ETH_RxDescListTypeDef::CurRxDesc`
- `uint32_t ETH_RxDescListTypeDef::FirstAppDesc`
- `uint32_t ETH_RxDescListTypeDef::AppDescNbr`
- `uint32_t ETH_RxDescListTypeDef::AppContextDesc`
- `uint32_t ETH_RxDescListTypeDef::ItMode`

21.1.5 ETH_RxPacketInfo

Data Fields

- `uint32_t SegmentCnt`
- `uint32_t VlanTag`
- `uint32_t InnerVlanTag`
- `uint32_t Checksum`
- `uint32_t HeaderType`
- `uint32_t PayloadType`
- `uint32_t MacFilterStatus`
- `uint32_t L3FilterStatus`
- `uint32_t L4FilterStatus`
- `uint32_t ErrorCode`

Field Documentation

- `uint32_t ETH_RxPacketInfo::SegmentCnt`
- `uint32_t ETH_RxPacketInfo::VlanTag`
- `uint32_t ETH_RxPacketInfo::InnerVlanTag`
- `uint32_t ETH_RxPacketInfo::Checksum`
- `uint32_t ETH_RxPacketInfo::HeaderType`
- `uint32_t ETH_RxPacketInfo::PayloadType`
- `uint32_t ETH_RxPacketInfo::MacFilterStatus`
- `uint32_t ETH_RxPacketInfo::L3FilterStatus`
- `uint32_t ETH_RxPacketInfo::L4FilterStatus`
- `uint32_t ETH_RxPacketInfo::ErrorCode`

21.1.6 ETH_MACConfigTypeDef

Data Fields

- `uint32_t SourceAddrControl`
- `FunctionalState ChecksumOffload`
- `uint32_t InterPacketGapVal`
- `FunctionalState GiantPacketSizeLimitControl`
- `FunctionalState Support2KPacket`
- `FunctionalState CRCStripTypePacket`

- **FunctionalState AutomaticPadCRCStrip**
- **FunctionalState Watchdog**
- **FunctionalState Jabber**
- **FunctionalState JumboPacket**
- **uint32_t Speed**
- **uint32_t DuplexMode**
- **FunctionalState LoopbackMode**
- **FunctionalState CarrierSenseBeforeTransmit**
- **FunctionalState ReceiveOwn**
- **FunctionalState CarrierSenseDuringTransmit**
- **FunctionalState RetryTransmission**
- **uint32_t BackOffLimit**
- **FunctionalState DeferralCheck**
- **uint32_t PreambleLength**
- **FunctionalState UnicastSlowProtocolPacketDetect**
- **FunctionalState SlowProtocolDetect**
- **FunctionalState CRCCheckingRxPackets**
- **uint32_t GiantPacketSizeLimit**
- **FunctionalState ExtendedInterPacketGap**
- **uint32_t ExtendedInterPacketGapVal**
- **FunctionalState ProgrammableWatchdog**
- **uint32_t WatchdogTimeout**
- **uint32_t PauseTime**
- **FunctionalState ZeroQuantaPause**
- **uint32_t PauseLowThreshold**
- **FunctionalState TransmitFlowControl**
- **FunctionalState UnicastPausePacketDetect**
- **FunctionalState ReceiveFlowControl**
- **uint32_t TransmitQueueMode**
- **uint32_t ReceiveQueueMode**
- **FunctionalState DropTCPIPChecksumErrorPacket**
- **FunctionalState ForwardRxErrorPacket**
- **FunctionalState ForwardRxUndersizedGoodPacket**

Field Documentation

- **uint32_t ETH_MACConfigTypeDef::SourceAddrControl**
Selects the Source Address Insertion or Replacement Control. This parameter can be a value of **ETH Source Addr Control**
- **FunctionalState ETH_MACConfigTypeDef::ChecksumOffload**
Enables or Disable the checksum checking for received packet payloads TCP, UDP or ICMP headers
- **uint32_t ETH_MACConfigTypeDef::InterPacketGapVal**
Sets the minimum IPG between Packet during transmission. This parameter can be a value of **ETH Inter Packet Gap**
- **FunctionalState ETH_MACConfigTypeDef::GiantPacketSizeLimitControl**
Enables or disables the Giant Packet Size Limit Control.
- **FunctionalState ETH_MACConfigTypeDef::Support2KPacket**
Enables or disables the IEEE 802.3as Support for 2K length Packets
- **FunctionalState ETH_MACConfigTypeDef::CRCStripTypePacket**

- Enables or disables the CRC stripping for Type packets.
- ***FunctionalState ETH_MACConfigTypeDef::AutomaticPadCRCStrip***
Enables or disables the Automatic MAC Pad/CRC Stripping.
- ***FunctionalState ETH_MACConfigTypeDef::Watchdog***
Enables or disables the Watchdog timer on Rx path When enabled, the MAC allows no more than 2048 bytes to be received. When disabled, the MAC can receive up to 16384 bytes.
- ***FunctionalState ETH_MACConfigTypeDef::Jabber***
Enables or disables Jabber timer on Tx path When enabled, the MAC allows no more than 2048 bytes to be sent. When disabled, the MAC can send up to 16384 bytes.
- ***FunctionalState ETH_MACConfigTypeDef::JumboPacket***
Enables or disables receiving Jumbo Packet When enabled, the MAC allows jumbo packets of 9,018 bytes without reporting a giant packet error
- ***uint32_t ETH_MACConfigTypeDef::Speed***
Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of ***ETH Speed***
- ***uint32_t ETH_MACConfigTypeDef::DuplexMode***
Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode This parameter can be a value of ***ETH Duplex Mode***
- ***FunctionalState ETH_MACConfigTypeDef::LoopbackMode***
Enables or disables the loopback mode
- ***FunctionalState ETH_MACConfigTypeDef::CarrierSenseBeforeTransmit***
Enables or disables the Carrier Sense Before Transmission in Full Duplex Mode.
- ***FunctionalState ETH_MACConfigTypeDef::ReceiveOwn***
Enables or disables the Receive Own in Half Duplex mode.
- ***FunctionalState ETH_MACConfigTypeDef::CarrierSenseDuringTransmit***
Enables or disables the Carrier Sense During Transmission in the Half Duplex mode
- ***FunctionalState ETH_MACConfigTypeDef::RetryTransmission***
Enables or disables the MAC retry transmission, when a collision occurs in Half Duplex mode.
- ***uint32_t ETH_MACConfigTypeDef::BackOffLimit***
Selects the BackOff limit value. This parameter can be a value of ***ETH Back Off Limit***
- ***FunctionalState ETH_MACConfigTypeDef::DeferralCheck***
Enables or disables the deferral check function in Half Duplex mode.
- ***uint32_t ETH_MACConfigTypeDef::PreambleLength***
Selects or not the Preamble Length for Transmit packets (Full Duplex mode). This parameter can be a value of ***ETH Preamble Length***
- ***FunctionalState ETH_MACConfigTypeDef::UnicastSlowProtocolPacketDetect***
Enable or disables the Detection of Slow Protocol Packets with unicast address.
- ***FunctionalState ETH_MACConfigTypeDef::SlowProtocolDetect***
Enable or disables the Slow Protocol Detection.
- ***FunctionalState ETH_MACConfigTypeDef::CRCCheckingRxPackets***
Enable or disables the CRC Checking for Received Packets.
- ***uint32_t ETH_MACConfigTypeDef::GiantPacketSizeLimit***

Specifies the packet size that the MAC will declare it as Giant, If its size is greater than the value programmed in this field in units of bytes This parameter must be a number between Min_Data = 0x618 (1518 byte) and Max_Data = 0xFFFF (32 Kbyte)

- **FunctionalState ETH_MACConfigTypeDef::ExtendedInterPacketGap**

Enable or disables the extended inter packet gap.

- **uint32_t ETH_MACConfigTypeDef::ExtendedInterPacketGapVal**

Sets the Extended IPG between Packet during transmission. This parameter can be a value from 0x0 to 0xFF

- **FunctionalState ETH_MACConfigTypeDef::ProgrammableWatchdog**

Enable or disables the Programmable Watchdog.

- **uint32_t ETH_MACConfigTypeDef::WatchdogTimeout**

This field is used as watchdog timeout for a received packet This parameter can be a value of **ETH Watchdog Timeout**

- **uint32_t ETH_MACConfigTypeDef::PauseTime**

This field holds the value to be used in the Pause Time field in the transmit control packet. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFF

- **FunctionalState ETH_MACConfigTypeDef::ZeroQuantaPause**

Enable or disables the automatic generation of Zero Quanta Pause Control packets.

- **uint32_t ETH_MACConfigTypeDef::PauseLowThreshold**

This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Packet. This parameter can be a value of **ETH Pause Low Threshold**

- **FunctionalState ETH_MACConfigTypeDef::TransmitFlowControl**

Enables or disables the MAC to transmit Pause packets in Full Duplex mode or the MAC back pressure operation in Half Duplex mode

- **FunctionalState ETH_MACConfigTypeDef::UnicastPausePacketDetect**

Enables or disables the MAC to detect Pause packets with unicast address of the station

- **FunctionalState ETH_MACConfigTypeDef::ReceiveFlowControl**

Enables or disables the MAC to decodes the received Pause packet and disables its transmitter for a specified (Pause) time

- **uint32_t ETH_MACConfigTypeDef::TransmitQueueMode**

Specifies the Transmit Queue operating mode. This parameter can be a value of **ETH Transmit Mode**

- **uint32_t ETH_MACConfigTypeDef::ReceiveQueueMode**

Specifies the Receive Queue operating mode. This parameter can be a value of **ETH Receive Mode**

- **FunctionalState ETH_MACConfigTypeDef::DropTCPIPChecksumErrorPacket**

Enables or disables Dropping of TCPIP Checksum Error Packets.

- **FunctionalState ETH_MACConfigTypeDef::ForwardRxErrorPacket**

Enables or disables forwarding Error Packets.

- **FunctionalState ETH_MACConfigTypeDef::ForwardRxUndersizedGoodPacket**

Enables or disables forwarding Undersized Good Packets.

21.1.7 ETH_DMAConfigTypeDef

Data Fields

- **uint32_t DMAArbitration**
- **FunctionalState AddressAlignedBeats**

- *uint32_t BurstMode*
- *FunctionalState RebuildINCRxBurst*
- *FunctionalState PBLx8Mode*
- *uint32_t TxDMABurstLength*
- *FunctionalState SecondPacketOperate*
- *uint32_t RxDMABurstLength*
- *FunctionalState FlushRxPacket*
- *FunctionalState TCPSegmentation*
- *uint32_t MaximumSegmentSize*

Field Documentation

- ***uint32_t ETH_DMAConfigTypeDef::DMAArbitration***
Sets the arbitration scheme between DMA Tx and Rx This parameter can be a value of ***ETH DMA Arbitration***
- ***FunctionalState ETH_DMAConfigTypeDef::AddressAlignedBeats***
Enables or disables the AHB Master interface address aligned burst transfers on Read and Write channels
- ***uint32_t ETH_DMAConfigTypeDef::BurstMode***
Sets the AHB Master interface burst transfers. This parameter can be a value of ***ETH Burst Mode***
- ***FunctionalState ETH_DMAConfigTypeDef::RebuildINCRxBurst***
Enables or disables the AHB Master to rebuild the pending beats of any initiated burst transfer with INCRx and SINGLE transfers.
- ***FunctionalState ETH_DMAConfigTypeDef::PBLx8Mode***
Enables or disables the PBL multiplication by eight.
- ***uint32_t ETH_DMAConfigTypeDef::TxDMABurstLength***
Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of ***ETH Tx DMA Burst Length***
- ***FunctionalState ETH_DMAConfigTypeDef::SecondPacketOperate***
Enables or disables the Operate on second Packet mode, which allows the DMA to process a second Packet of Transmit data even before obtaining the status for the first one.
- ***uint32_t ETH_DMAConfigTypeDef::RxDMABurstLength***
Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of ***ETH Rx DMA Burst Length***
- ***FunctionalState ETH_DMAConfigTypeDef::FlushRxPacket***
Enables or disables the Rx Packet Flush
- ***FunctionalState ETH_DMAConfigTypeDef::TCPSegmentation***
Enables or disables the TCP Segmentation
- ***uint32_t ETH_DMAConfigTypeDef::MaximumSegmentSize***
Sets the maximum segment size that should be used while segmenting the packet This parameter can be a value from 0x40 to 0x3FFF

21.1.8 ETH_InitTypeDef

Data Fields

- *uint8_t * MACAddr*
- *ETH_MedialInterfaceTypeDef MedialInterface*
- *ETH_DMADescTypeDef * TxDesc*
- *ETH_DMADescTypeDef * RxDesc*

- `uint32_t RxBuffLen`

Field Documentation

- `uint8_t* ETH_InitTypeDef::MACAddr`

MAC Address of used Hardware: must be pointer on an array of 6 bytes

- `ETH_MedialInterfaceTypeDef ETH_InitTypeDef::MedialInterface`

Selects the MII interface or the RMII interface.

- `ETH_DMADescTypeDef* ETH_InitTypeDef::TxDesc`

Provides the address of the first DMA Tx descriptor in the list

- `ETH_DMADescTypeDef* ETH_InitTypeDef::RxDesc`

Provides the address of the first DMA Rx descriptor in the list

- `uint32_t ETH_InitTypeDef::RxBuffLen`

Provides the length of Rx buffers size

21.1.9 ETH_HandleTypeDef

Data Fields

- `ETH_TypeDef * Instance`
- `ETH_InitTypeDef Init`
- `ETH_TxDescListTypeDef TxDescList`
- `ETH_RxDescListTypeDef RxDescList`
- `HAL_LockTypeDef Lock`
- `__IO HAL_ETH_StateTypeDef gState`
- `__IO HAL_ETH_StateTypeDef RxState`
- `__IO uint32_t ErrorCode`
- `__IO uint32_t DMAErrorCode`
- `__IO uint32_t MACErrorCode`
- `__IO uint32_t MACWakeUpEvent`
- `__IO uint32_t MACLPIEvent`

Field Documentation

- `ETH_TypeDef* ETH_HandleTypeDef::Instance`

Register base address

- `ETH_InitTypeDef ETH_HandleTypeDef::Init`

Ethernet Init Configuration

- `ETH_TxDescListTypeDef ETH_HandleTypeDef::TxDescList`

Tx descriptor wrapper: holds all Tx descriptors list addresses and current descriptor index

- `ETH_RxDescListTypeDef ETH_HandleTypeDef::RxDescList`

Rx descriptor wrapper: holds all Rx descriptors list addresses and current descriptor index

- `HAL_LockTypeDef ETH_HandleTypeDef::Lock`

Locking object

- `__IO HAL_ETH_StateTypeDef ETH_HandleTypeDef::gState`

ETH state information related to global Handle management and also related to Tx operations. This parameter can be a value of `HAL_ETH_StateTypeDef`

- `__IO HAL_ETH_StateTypeDef ETH_HandleTypeDef::RxState`

- ETH state information related to Rx operations. This parameter can be a value of **HAL_ETHERNET_StatusTypeDef**
- **`__IO uint32_t ETH_HandleTypeDef::ErrorCode`**
Holds the global Error code of the ETH HAL status machine This parameter can be a value of **ETH_Error_Code**
 - **`__IO uint32_t ETH_HandleTypeDef::DMAErrorCode`**
Holds the DMA Rx Tx Error code when a DMA AIS interrupt occurs This parameter can be a combination of **ETH_DMA_Status_Flags**
 - **`__IO uint32_t ETH_HandleTypeDef::MACErrorCode`**
Holds the MAC Rx Tx Error code when a MAC Rx or Tx status interrupt occurs This parameter can be a combination of **ETH_MAC_Rx_Tx_Status**
 - **`__IO uint32_t ETH_HandleTypeDef::MACWakeUpEvent`**
Holds the Wake Up event when the MAC exit the power down mode This parameter can be a value of **ETH_MAC_Wake_Up_Event**
 - **`__IO uint32_t ETH_HandleTypeDef::MACLPIEvent`**
Holds the LPI event when the an LPI status interrupt occurs. This parameter can be a value of **ETH_Ex_LPI_Event**

21.1.10 ETH_MACFilterConfigTypeDef

Data Fields

- **`FunctionalState PromiscuousMode`**
- **`FunctionalState ReceiveAllMode`**
- **`FunctionalState HashOrPerfectFilter`**
- **`FunctionalState HashUnicast`**
- **`FunctionalState HashMulticast`**
- **`FunctionalState PassAllMulticast`**
- **`FunctionalState SrcAddrFiltering`**
- **`FunctionalState SrcAddrInverseFiltering`**
- **`FunctionalState DestAddrInverseFiltering`**
- **`FunctionalState BroadcastFilter`**
- **`uint32_t ControlPacketsFilter`**

Field Documentation

- **`FunctionalState ETH_MACFilterConfigTypeDef::PromiscuousMode`**
Enable or Disable Promiscuous Mode
- **`FunctionalState ETH_MACFilterConfigTypeDef::ReceiveAllMode`**
Enable or Disable Receive All Mode
- **`FunctionalState ETH_MACFilterConfigTypeDef::HashOrPerfectFilter`**
Enable or Disable Perfect filtering in addition to Hash filtering
- **`FunctionalState ETH_MACFilterConfigTypeDef::HashUnicast`**
Enable or Disable Hash filtering on unicast packets
- **`FunctionalState ETH_MACFilterConfigTypeDef::HashMulticast`**
Enable or Disable Hash filtering on multicast packets
- **`FunctionalState ETH_MACFilterConfigTypeDef::PassAllMulticast`**
Enable or Disable passing all multicast packets
- **`FunctionalState ETH_MACFilterConfigTypeDef::SrcAddrFiltering`**

- Enable or Disable source address filtering module
- ***FunctionalState ETH_MACFilterConfigTypeDef::SrcAddrInverseFiltering***
Enable or Disable source address inverse filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::DestAddrInverseFiltering***
Enable or Disable destination address inverse filtering
- ***FunctionalState ETH_MACFilterConfigTypeDef::BroadcastFilter***
Enable or Disable broadcast filter
- ***uint32_t ETH_MACFilterConfigTypeDef::ControlPacketsFilter***
Set the control packets filter This parameter can be a value of ***ETH Control Packets Filter***

21.1.11 ETH_PowerDownConfigTypeDef

Data Fields

- ***FunctionalState WakeUpPacket***
- ***FunctionalState MagicPacket***
- ***FunctionalState GlobalUnicast***
- ***FunctionalState WakeUpForward***

Field Documentation

- ***FunctionalState ETH_PowerDownConfigTypeDef::WakeUpPacket***
Enable or Disable Wake up packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::MagicPacket***
Enable or Disable Magic packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::GlobalUnicast***
Enable or Disable Global unicast packet detection in power down mode
- ***FunctionalState ETH_PowerDownConfigTypeDef::WakeUpForward***
Enable or Disable Forwarding Wake up packets

21.2 ETH Firmware driver API description

21.2.1 How to use this driver

The ETH HAL driver can be used as follows:

1. Declare a ETH_HandleTypeDef handle structure, for example: ETH_HandleTypeDef heth;
2. Fill parameters of Init structure in heth handle
3. Call HAL_ETH_Init() API to initialize the Ethernet peripheral (MAC, DMA, ...)
4. Initialize the ETH low level resources through the HAL_ETH_MsplInit() API:
 - a. Enable the Ethernet interface clock using
 - __HAL_RCC_ETH1MAC_CLK_ENABLE()
 - __HAL_RCC_ETH1TX_CLK_ENABLE()
 - __HAL_RCC_ETH1RX_CLK_ENABLE()
 - b. Initialize the related GPIO clocks
 - c. Configure Ethernet pinout
 - d. Configure Ethernet NVIC interrupt (in Interrupt mode)
5. Ethernet data reception is asynchronous, so call the following API to start the listening mode:

- a. HAL_ETH_Start(): This API starts the MAC and DMA transmission and reception process, without enabling end of transfer interrupts, in this mode user has to poll for data availability by calling HAL_ETH_IsRxDataAvailable()
 - b. HAL_ETH_Start_IT(): This API starts the MAC and DMA transmission and reception process, end of transfer interrupts are enabled in this mode, HAL_ETH_RxCpltCallback() will be executed when an Ethernet packet is received
6. When data is received (HAL_ETH_IsRxDataAvailable() returns 1 or Rx interrupt occurred), user can call the following APIs to get received data:
 - a. HAL_ETH_GetRxDataBuffer(): Get buffer address of received frame
 - b. HAL_ETH_GetRxDataLength(): Get received frame length
 - c. HAL_ETH_GetRxDataInfo(): Get received frame additional info, please refer to ETH_RxPacketInfo typedef structure
 7. For transmission path, two APIs are available:
 - a. HAL_ETH_Transmit(): Transmit an ETH frame in blocking mode
 - b. HAL_ETH_Transmit_IT(): Transmit an ETH frame in interrupt mode, HAL_ETH_TxCpltCallback() will be executed when end of transfer occur
 8. Communication with an external PHY device:
 - a. HAL_ETH_ReadPHYRegister(): Read a register from an external PHY
 - b. HAL_ETH_WritePHYRegister(): Write data to an external RHY register
 9. Configure the Ethernet MAC after ETH peripheral initialization
 - a. HAL_ETH_GetMACConfig(): Get MAC actual configuration into ETH_MACConfigTypeDef
 - b. HAL_ETH_SetMACConfig(): Set MAC configuration based on ETH_MACConfigTypeDef
 10. Configure the Ethernet DMA after ETH peripheral initialization
 - a. HAL_ETH_GetDMAConfig(): Get DMA actual configuration into ETH_DMAConfigTypeDef
 - b. HAL_ETH_SetDMAConfig(): Set DMA configuration based on ETH_DMAConfigTypeDef -@- The PTP protocol offload APIs are not supported in this driver.

21.2.2

Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize and deinitialize the ETH peripheral:

- User must Implement HAL_ETH_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO and NVIC).
- Call the function HAL_ETH_Init() to configure the selected device with the selected configuration:
 - MAC address
 - Media interface (MII or RMII)
 - Rx DMA Descriptors Tab
 - Tx DMA Descriptors Tab
 - Length of Rx Buffers
- Call the function HAL_ETH_DescAssignMemory() to assign data buffers for each Rx DMA Descriptor
- Call the function HAL_ETH_DelInit() to restore the default configuration of the selected ETH peripheral.

This section contains the following APIs:

- [**HAL_ETH_Init**](#)
- [**HAL_ETH_DelInit**](#)
- [**HAL_ETH_MspInit**](#)
- [**HAL_ETH_MspDelInit**](#)
- [**HAL_ETH_DescAssignMemory**](#)

21.2.3

IO operation functions

This subsection provides a set of functions allowing to manage the ETH data transfer.

This section contains the following APIs:

- [`HAL_ETH_Start`](#)
- [`HAL_ETH_Start_IT`](#)
- [`HAL_ETH_Stop`](#)
- [`HAL_ETH_Stop_IT`](#)
- [`HAL_ETH_Transmit`](#)
- [`HAL_ETH_Transmit_IT`](#)
- [`HAL_ETH_IsRxDataAvailable`](#)
- [`HAL_ETH_GetRxDataBuffer`](#)
- [`HAL_ETH_GetRxDataLength`](#)
- [`HAL_ETH_GetRxDataInfo`](#)
- [`HAL_ETH_BuildRxDescriptors`](#)
- [`HAL_ETH_IRQHandler`](#)
- [`HAL_ETH_TxCpltCallback`](#)
- [`HAL_ETH_RxCpltCallback`](#)
- [`HAL_ETH_DMAErrorCallback`](#)
- [`HAL_ETH_MACErrorCallback`](#)
- [`HAL_ETH_PMTCallback`](#)
- [`HAL_ETH_EEECallback`](#)
- [`HAL_ETH_WakeUpCallback`](#)
- [`HAL_ETH_ReadPHYRegister`](#)
- [`HAL_ETH_WritePHYRegister`](#)

21.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the ETH peripheral.

This section contains the following APIs:

- [`HAL_ETH_GetMACConfig`](#)
- [`HAL_ETH_SetMACConfig`](#)
- [`HAL_ETH_SetDMAConfig`](#)
- [`HAL_ETH_SetMDIOClockRange`](#)
- [`HAL_ETH_SetMACFilterConfig`](#)
- [`HAL_ETH_GetMACFilterConfig`](#)
- [`HAL_ETH_SetSourceMACAddrMatch`](#)
- [`HAL_ETH_SetHashTable`](#)
- [`HAL_ETH_SetRxVLANIdentifier`](#)
- [`HAL_ETH_EnterPowerDownMode`](#)
- [`HAL_ETH_ExitPowerDownMode`](#)
- [`HAL_ETH_SetWakeUpFilter`](#)

21.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of ETH communication process, return Peripheral Errors occurred during communication process

This section contains the following APIs:

- [`HAL_ETH_GetState`](#)
- [`HAL_ETH_GetError`](#)
- [`HAL_ETH_GetDMAError`](#)
- [`HAL_ETH_GetMACError`](#)
- [`HAL_ETH_GetMACWakeUpSource`](#)

21.2.6 Detailed description of functions

HAL_ETH_Init

Function name

`HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)`

Function description

Initialize the Ethernet peripheral registers.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_DeInit

Function name

`HAL_StatusTypeDef HAL_ETH_DeInit (ETH_HandleTypeDef * heth)`

Function description

DeInitializes the ETH peripheral.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_MspInit

Function name

`void HAL_ETH_MspInit (ETH_HandleTypeDef * heth)`

Function description

Initializes the ETH MSP.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_MspDeInit

Function name

`void HAL_ETH_MspDeInit (ETH_HandleTypeDef * heth)`

Function description

DeInitializes ETH MSP.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_DescAssignMemory

Function name

HAL_StatusTypeDef HAL_ETH_DescAssignMemory (ETH_HandleTypeDef * heth, uint32_t Index, uint8_t * pBuffer1, uint8_t * pBuffer2)

Function description

Assign memory buffers to a DMA Rx descriptor.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Index:** : index of the DMA Rx descriptor this parameter can be a value from 0x0 to (ETH_RX_DESC_CNT -1)
- **pBuffer1:** address of buffer1
- **pBuffer2:** address of buffer 2 if available

Return values

- **HAL:** status

HAL_ETH_Start

Function name

HAL_StatusTypeDef HAL_ETH_Start (ETH_HandleTypeDef * heth)

Function description

Enables Ethernet MAC and DMA reception and transmission.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Start_IT

Function name

HAL_StatusTypeDef HAL_ETH_Start_IT (ETH_HandleTypeDef * heth)

Function description

Enables Ethernet MAC and DMA reception/transmission in Interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Stop

Function name

`HAL_StatusTypeDef HAL_ETH_Stop (ETH_HandleTypeDef * heth)`

Function description

Stop Ethernet MAC and DMA reception/transmission.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_Stop_IT

Function name

`HAL_StatusTypeDef HAL_ETH_Stop_IT (ETH_HandleTypeDef * heth)`

Function description

Stop Ethernet MAC and DMA reception/transmission in Interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_IsRxDataAvailable

Function name

`uint8_t HAL_ETH_IsRxDataAvailable (ETH_HandleTypeDef * heth)`

Function description

Checks for received Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **1:** A Packet is received 0: no Packet received

HAL_ETH_GetRxDataBuffer

Function name

`HAL_StatusTypeDef HAL_ETH_GetRxDataBuffer (ETH_HandleTypeDef * heth, ETH_BufferTypeDef * RxBuffer)`

Function description

This function gets the buffer address of last received Packet.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

- **RxBuffer:** Pointer to a ETH_BufferTypeDef structure

Return values

- **HAL:** status

HAL_ETH_GetRxDataLength

Function name

HAL_StatusTypeDef HAL_ETH_GetRxDataLength (ETH_HandleTypeDef * heth, uint32_t * Length)

Function description

This function gets the length of last received Packet.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Length:** parameter to hold Rx packet length

Return values

- **HAL:** Status

HAL_ETH_GetRxDataInfo

Function name

HAL_StatusTypeDef HAL_ETH_GetRxDataInfo (ETH_HandleTypeDef * heth, ETH_RxPacketInfo * RxPacketInfo)

Function description

Get the Rx data info (Packet type, VLAN tag, Filters status, ...)

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **RxPacketInfo:** parameter to hold info of received buffer

Return values

- **HAL:** status

HAL_ETH_BuildRxDescriptors

Function name

HAL_StatusTypeDef HAL_ETH_BuildRxDescriptors (ETH_HandleTypeDef * heth)

Function description

This function gives back Rx Desc of the last received Packet to the DMA, so ETH DMA will be able to use these descriptors to receive next Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status.

HAL_ETHERNET_Transmit

Function name

```
HAL_StatusTypeDef HAL_ETHERNET_Transmit (ETH_HandleTypeDef * heth, ETH_TxPacketConfig * pTxConfig,  
uint32_t Timeout)
```

Function description

Sends an Ethernet Packet in polling mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pTxConfig:** Hold the configuration of packet to be transmitted
- **Timeout:** timeout value

Return values

- **HAL:** status

HAL_ETHERNET_Transmit_IT

Function name

```
HAL_StatusTypeDef HAL_ETHERNET_Transmit_IT (ETH_HandleTypeDef * heth, ETH_TxPacketConfig *  
pTxConfig)
```

Function description

Sends an Ethernet Packet in interrupt mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pTxConfig:** Hold the configuration of packet to be transmitted

Return values

- **HAL:** status

HAL_ETHERNET_WritePHYRegister

Function name

```
HAL_StatusTypeDef HAL_ETHERNET_WritePHYRegister (ETH_HandleTypeDef * heth, uint32_t PHYAddr,  
uint32_t PHYReg, uint32_t RegValue)
```

Function description

Writes to a PHY register.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **PHYAddr:** PHY port address, must be a value from 0 to 31
- **PHYReg:** PHY register address, must be a value from 0 to 31
- **RegValue:** the value to write

Return values

- **HAL:** status

HAL_ETH_ReadPHYRegister

Function name

```
HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint32_t PHYAddr,  
uint32_t PHYReg, uint32_t * pRegValue)
```

Function description

Read a PHY register.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **PHYAddr:** PHY port address, must be a value from 0 to 31
- **PHYReg:** PHY register address, must be a value from 0 to 31
- **pRegValue:** parameter to hold read value

Return values

- **HAL:** status

HAL_ETH_IRQHandler

Function name

```
void HAL_ETH_IRQHandler (ETH_HandleTypeDef * heth)
```

Function description

This function handles ETH interrupt request.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** status

HAL_ETH_TxCpltCallback

Function name

```
void HAL_ETH_TxCpltCallback (ETH_HandleTypeDef * heth)
```

Function description

Tx Transfer completed callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_RxCpltCallback

Function name

```
void HAL_ETH_RxCpltCallback (ETH_HandleTypeDef * heth)
```

Function description

Rx Transfer completed callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_DMAErrorCallback

Function name

```
void HAL_ETH_DMAErrorCallback (ETH_HandleTypeDef * heth)
```

Function description

Ethernet DMA transfer error callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_MACErrorCallback

Function name

```
void HAL_ETH_MACErrorCallback (ETH_HandleTypeDef * heth)
```

Function description

Ethernet MAC transfer error callbacks.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_PMTCallback

Function name

```
void HAL_ETH_PMTCallback (ETH_HandleTypeDef * heth)
```

Function description

Ethernet Power Management module IT callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_EEECallback

Function name

```
void HAL_ETH_EEECallback (ETH_HandleTypeDef * heth)
```

Function description

Energy Efficient Etherent IT callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_WakeUpCallback

Function name

void HAL_ETH_WakeUpCallback (ETH_HandleTypeDef * heth)

Function description

ETH WAKEUP interrupt callback.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETH_GetMACConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetMACConfig (ETH_HandleTypeDef * heth, ETH_MACConfigTypeDef * macconf)

Function description

Get the configuration of the MAC and MTL subsystems.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **macconf:** pointer to a ETH_MACConfigTypeDef structure that will hold the configuration of the MAC.

Return values

- **HAL:** Status

HAL_ETH_GetDMAConfig

Function name

HAL_StatusTypeDef HAL_ETH_GetDMAConfig (ETH_HandleTypeDef * heth, ETH_DMAConfigTypeDef * dmaconf)

Function description

Get the configuration of the DMA.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **dmaconf:** pointer to a ETH_DMAConfigTypeDef structure that will hold the configuration of the ETH DMA.

Return values

- **HAL:** Status

HAL_ETHERNET_SetMACConfig

Function name

```
HAL_StatusTypeDef HAL_ETHERNET_SetMACConfig (ETH_HandleTypeDef * heth, ETH_MACConfigTypeDef * macconf)
```

Function description

Set the MAC configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **macconf:** pointer to a ETH_MACConfigTypeDef structure that contains the configuration of the MAC.

Return values

- **HAL:** status

HAL_ETHERNET_SetDMAConfig

Function name

```
HAL_StatusTypeDef HAL_ETHERNET_SetDMAConfig (ETH_HandleTypeDef * heth, ETH_DMAConfigTypeDef * dmaconf)
```

Function description

Set the ETH DMA configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **dmaconf:** pointer to a ETH_DMAConfigTypeDef structure that will hold the configuration of the ETH DMA.

Return values

- **HAL:** status

HAL_ETHERNET_SetMDIOPortClockRange

Function name

```
void HAL_ETHERNET_SetMDIOPortClockRange (ETH_HandleTypeDef * heth)
```

Function description

Configures the Clock range of ETH MDIO interface.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHERNET_SetRxVLANIdentifier

Function name

```
void HAL_ETHERNET_SetRxVLANIdentifier (ETH_HandleTypeDef * heth, uint32_t ComparisonBits, uint32_t VLANIdentifier)
```

Function description

Set the VLAN Identifier for Rx packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **ComparisonBits:** 12 or 16 bit comparison mode must be a value of ETH VLAN Tag Comparison
- **VLANIdentifier:** VLAN Identifier value

Return values

- **None:**

HAL_ETH_GetMACFilterConfig

Function name

**HAL_StatusTypeDef HAL_ETH_GetMACFilterConfig (ETH_HandleTypeDef * heth,
ETH_MACFilterConfigTypeDef * pFilterConfig)**

Function description

Get the ETH MAC (L2) Filters configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilterConfig:** pointer to a ETH_MACFilterConfigTypeDef structure that will hold the configuration of the ETH MAC filters.

Return values

- **HAL:** status

HAL_ETH_SetMACFilterConfig

Function name

**HAL_StatusTypeDef HAL_ETH_SetMACFilterConfig (ETH_HandleTypeDef * heth,
ETH_MACFilterConfigTypeDef * pFilterConfig)**

Function description

Set the ETH MAC (L2) Filters configuration.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilterConfig:** pointer to a ETH_MACFilterConfigTypeDef structure that contains the configuration of the ETH MAC filters.

Return values

- **HAL:** status

HAL_ETH_SetHashTable

Function name

HAL_StatusTypeDef HAL_ETH_SetHashTable (ETH_HandleTypeDef * heth, uint32_t * pHഷTable)

Function description

Set the ETH Hash Table Value.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pHashTable:** pointer to a table of two 32 bit values, that contains the 64 bits of the hash table.

Return values

- **HAL:** status

HAL_ETH_SetSourceMACAddrMatch

Function name

```
HAL_StatusTypeDef HAL_ETH_SetSourceMACAddrMatch (ETH_HandleTypeDef * heth, uint32_t AddrNbr,  
uint8_t * pMACAddr)
```

Function description

Set the source MAC Address to be matched.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **AddrNbr:** The MAC address to configure This parameter must be a value of the following:
ETH_MAC_ADDRESS1 ETH_MAC_ADDRESS2 ETH_MAC_ADDRESS3
- **pMACAddr:** Pointer to MAC address buffer data (6 bytes)

Return values

- **HAL:** status

HAL_ETH_EnterPowerDownMode

Function name

```
void HAL_ETH_EnterPowerDownMode (ETH_HandleTypeDef * heth, ETH_PowerDownConfigTypeDef *  
pPowerDownConfig)
```

Function description

Enters the Power down mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pPowerDownConfig:** a pointer to ETH_PowerDownConfigTypeDef structure that contains the Power Down configuration

Return values

- **None.:**

HAL_ETH_ExitPowerDownMode

Function name

```
void HAL_ETH_ExitPowerDownMode (ETH_HandleTypeDef * heth)
```

Function description

Exits from the Power down mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETH_SetWakeUpFilter

Function name

HAL_StatusTypeDef HAL_ETH_SetWakeUpFilter (ETH_HandleTypeDef * heth, uint32_t * pFilter, uint32_t Count)

Function description

Set the WakeUp filter.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pFilter:** pointer to filter registers values
- **Count:** number of filter registers, must be from 1 to 8.

Return values

- **None.:**

HAL_ETH_GetState

Function name

HAL_ETH_StateTypeDef HAL_ETH_GetState (ETH_HandleTypeDef * heth)

Function description

Returns the ETH state.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **HAL:** state

HAL_ETH_GetError

Function name

uint32_t HAL_ETH_GetError (ETH_HandleTypeDef * heth)

Function description

Returns the ETH error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** Error Code

HAL_ETH_GetDMAError

Function name

uint32_t HAL_ETH_GetDMAError (ETH_HandleTypeDef * heth)

Function description

Returns the ETH DMA error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** DMA Error Code

HAL_ETH_GetMACError

Function name

uint32_t HAL_ETH_GetMACError (ETH_HandleTypeDef * heth)

Function description

Returns the ETH MAC error code.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC Error Code

HAL_ETH_GetMACWakeUpSource

Function name

uint32_t HAL_ETH_GetMACWakeUpSource (ETH_HandleTypeDef * heth)

Function description

Returns the ETH MAC WakeUp event source.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC WakeUp event source

21.3 ETH Firmware driver defines

21.3.1 ETH

ETH Back Off Limit

ETH_BACKOFFLIMIT_10

ETH_BACKOFFLIMIT_8

ETH_BACKOFFLIMIT_4

ETH_BACKOFFLIMIT_1

ETH Burst Mode

ETH_BURSTLENGTH_FIXED

ETH_BURSTLENGTH_MIXED

ETH_BURSTLENGTH_UNSPECIFIED

ETH Control Packets Filter

ETH_CTRLPACKETS_BLOCK_ALL

ETH_CTRLPACKETS_FORWARD_ALL_EXCEPT_PA

ETH_CTRLPACKETS_FORWARD_ALL

ETH_CTRLPACKETS_FORWARD_PASSED_ADDR_FILTER

ETH DMA Arbitration

ETH_DMAARBITRATION_RX

ETH_DMAARBITRATION_RX1_TX1

ETH_DMAARBITRATION_RX2_TX1

ETH_DMAARBITRATION_RX3_TX1

ETH_DMAARBITRATION_RX4_TX1

ETH_DMAARBITRATION_RX5_TX1

ETH_DMAARBITRATION_RX6_TX1

ETH_DMAARBITRATION_RX7_TX1

ETH_DMAARBITRATION_RX8_TX1

ETH_DMAARBITRATION_TX

ETH_DMAARBITRATION_TX1_RX1

ETH_DMAARBITRATION_TX2_RX1

ETH_DMAARBITRATION_TX3_RX1

ETH_DMAARBITRATION_TX4_RX1

ETH_DMAARBITRATION_TX5_RX1

ETH_DMAARBITRATION_TX6_RX1

ETH_DMAARBITRATION_TX7_RX1

ETH_DMAARBITRATION_TX8_RX1

ETH DMA Interrupts

ETH_DMA_NORMAL_IT

ETH_DMA_ABNORMAL_IT
ETH_DMA_CONTEXT_DESC_ERROR_IT
ETH_DMA_FATAL_BUS_ERROR_IT
ETH_DMA_EARLY_RX_IT
ETH_DMA_EARLY_TX_IT
ETH_DMA_RX_WATCHDOG_TIMEOUT_IT
ETH_DMA_RX_PROCESS_STOPPED_IT
ETH_DMA_RX_BUFFER_UNAVAILABLE_IT
ETH_DMA_RX_IT
ETH_DMA_TX_BUFFER_UNAVAILABLE_IT
ETH_DMA_TX_PROCESS_STOPPED_IT
ETH_DMA_TX_IT

ETH DMA Rx Descriptor Bit Definition

ETH_DMARXNDESCRF_BUF1AP

Header or Buffer 1 Address Pointer

ETH_DMARXNDESCRF_BUF2AP

Buffer 2 Address Pointer

ETH_DMARXNDESCRF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMARXNDESCRF_IOC

Interrupt Enabled on Completion

ETH_DMARXNDESCRF_BUF2V

Buffer 2 Address Valid

ETH_DMARXNDESCRF_BUF1V

Buffer 1 Address Valid

ETH_DMARXNDESCWBF_IVT

Inner VLAN Tag

ETH_DMARXNDESCWBF_OVT

Outer VLAN Tag

ETH_DMARXNDESCWBF_OPC

OAM Sub-Type Code, or MAC Control Packet opcode

ETH_DMARXNDESCWBF_TD

Timestamp Dropped

ETH_DMARXNDESCWBF_TSA

Timestamp Available

ETH_DMARXNDESCWBF_PV

PTP Version

ETH_DMARXNDESCWBF_PFT

PTP Packet Type

ETH_DMARXNDESCWBF_PMT_NO

PTP Message Type: No PTP message received

ETH_DMARXNDESCWBF_PMT_SYNC

PTP Message Type: SYNC (all clock types)

ETH_DMARXNDESCWBF_PMT_FUP

PTP Message Type: Follow_Up (all clock types)

ETH_DMARXNDESCWBF_PMT_DREQ

PTP Message Type: Delay_Req (all clock types)

ETH_DMARXNDESCWBF_PMT_DRESP

PTP Message Type: Delay_Resp (all clock types)

ETH_DMARXNDESCWBF_PMT_PDREQ

PTP Message Type: Pdelay_Req (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_PDRESP

PTP Message Type: Pdelay_Resp (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_PDRESPFUP

PTP Message Type: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock)

ETH_DMARXNDESCWBF_PMT_ANNOUNCE

PTP Message Type: Announce

ETH_DMARXNDESCWBF_PMT_MANAG

PTP Message Type: Management

ETH_DMARXNDESCWBF_PMT_SIGN

PTP Message Type: Signaling

ETH_DMARXNDESCWBF_PMT_RESERVED

PTP Message Type: PTP packet with Reserved message type

ETH_DMARXNDESCWBF_IPCE

IP Payload Error

ETH_DMARXNDESCWBF_IPCB

IP Checksum Bypassed

ETH_DMARXNDESCWBF_IPV6

IPv6 header Present

ETH_DMARXNDESCWBF_IPV4

IPv4 header Present

ETH_DMARXNDESCWBF_IPHE

IP Header Error

ETH_DMARXNDESCWBF_PT

Payload Type mask

ETH_DMARXNDESCWBF_PT_UNKNOWN

Payload Type: Unknown type or IP/AV payload not processed

ETH_DMARXNDESCWBF_PT_UDP

Payload Type: UDP

ETH_DMARXNDESCWBF_PT_TCP

Payload Type: TCP

ETH_DMARXNDESCWBF_PT_ICMP

Payload Type: ICMP

ETH_DMARXNDESCWBF_L3L4FM

L3 and L4 Filter Number Matched: if reset filter 0 is matched , if set filter 1 is matched

ETH_DMARXNDESCWBF_L4FM

Layer 4 Filter Match

ETH_DMARXNDESCWBF_L3FM

Layer 3 Filter Match

ETH_DMARXNDESCWBF_MADRM

MAC Address Match or Hash Value

ETH_DMARXNDESCWBF_HF

Hash Filter Status

ETH_DMARXNDESCWBF_DAF

Destination Address Filter Fail

ETH_DMARXNDESCWBF_SAF

SA Address Filter Fail

ETH_DMARXNDESCWBF_VF

VLAN Filter Status

ETH_DMARXNDESCWBF_ARPNR

ARP Reply Not Generated

ETH_DMARXNDESCWBF_OWN

Own Bit

ETH_DMARXNDESCWBF_CTXT

Receive Context Descriptor

ETH_DMARXNDESCWBF_FD

First Descriptor

ETH_DMARXNDESCWBF_LD

Last Descriptor

ETH_DMARXNDESCWBF_RS2V

Receive Status RDES2 Valid

ETH_DMARXNDESCWBF_RS1V

Receive Status RDES1 Valid

ETH_DMARXNDESCWBF_RS0V

Receive Status RDES0 Valid

ETH_DMARXNDESCWBF_CE

CRC Error

ETH_DMARXNDESCWBF_GP

Giant Packet

ETH_DMARXNDESCWBF_RWT

Receive Watchdog Timeout

ETH_DMARXNDESCWBF_OE

Overflow Error

ETH_DMARXNDESCWBF_RE

Receive Error

ETH_DMARXNDESCWBF_DE

Dribble Bit Error

ETH_DMARXNDESCWBF_LT

Length/Type Field

ETH_DMARXNDESCWBF_LT_LP

The packet is a length packet

ETH_DMARXNDESCWBF_LT_TP

The packet is a type packet

ETH_DMARXNDESCWBF_LT_ARP

The packet is a ARP Request packet type

ETH_DMARXNDESCWBF_LT_VLAN

The packet is a type packet with VLAN Tag

ETH_DMARXNDESCWBF_LT_DVLAN

The packet is a type packet with Double VLAN Tag

ETH_DMARXNDESCWBF_LT_MAC

The packet is a MAC Control packet type

ETH_DMARXNDESCWBF_LT_OAM

The packet is a OAM packet type

ETH_DMARXNDESCWBF_ES

Error Summary

ETH_DMARXNDESCWBF_PL

Packet Length

ETH_DMARXCDESC_RTSL

Receive Packet Timestamp Low

ETH_DMARXCDESC_RTSH

Receive Packet Timestamp High

ETH_DMARXCDESC_OWN

Own Bit

ETH_DMARXCDESC_CTXT

Receive Context Descriptor

ETH DMA Status Flags**ETH_DMA_RX_NO_ERROR_FLAG****ETH_DMA_RX_DESC_READ_ERROR_FLAG****ETH_DMA_RX_DESC_WRITE_ERROR_FLAG****ETH_DMA_RX_BUFFER_READ_ERROR_FLAG****ETH_DMA_RX_BUFFER_WRITE_ERROR_FLAG****ETH_DMA_TX_NO_ERROR_FLAG****ETH_DMA_TX_DESC_READ_ERROR_FLAG****ETH_DMA_TX_DESC_WRITE_ERROR_FLAG****ETH_DMA_TX_BUFFER_READ_ERROR_FLAG****ETH_DMA_TX_BUFFER_WRITE_ERROR_FLAG****ETH_DMA_CONTEXT_DESC_ERROR_FLAG****ETH_DMA_FATAL_BUS_ERROR_FLAG****ETH_DMA_EARLY_TX_IT_FLAG****ETH_DMA_RX_WATCHDOG_TIMEOUT_FLAG****ETH_DMA_RX_PROCESS_STOPPED_FLAG****ETH_DMA_RX_BUFFER_UNAVAILABLE_FLAG****ETH_DMA_TX_PROCESS_STOPPED_FLAG*****ETH DMA Tx Descriptor Bit Definition*****ETH_DMATXNDESCRF_B1AP**

Transmit Packet Timestamp Low

ETH_DMATXNDESCRF_B2AP

Transmit Packet Timestamp High

ETH_DMATXNDESCRF_IOC

Interrupt on Completion

ETH_DMATXNDESCRF_TTSE

Transmit Timestamp Enable

ETH_DMATXNDESCRF_B2L

Buffer 2 Length

ETH_DMATXNDESCRF_VTIR

VLAN Tag Insertion or Replacement mask

ETH_DMATXNDESCRF_VTIR_DISABLE

Do not add a VLAN tag.

ETH_DMATXNDESCRF_VTIR_REMOVE

Remove the VLAN tag from the packets before transmission.

ETH_DMATXNDESCRF_VTIR_INSERT

Insert a VLAN tag.

ETH_DMATXNDESCRF_VTIR_REPLACE

Replace the VLAN tag.

ETH_DMATXNDESCRF_B1L

Buffer 1 Length

ETH_DMATXNDESCRF_HL

Header Length

ETH_DMATXNDESCRF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXNDESCRF_CTXT

Context Type

ETH_DMATXNDESCRF_FD

First Descriptor

ETH_DMATXNDESCRF_LD

Last Descriptor

ETH_DMATXNDESCRF_CPC

CRC Pad Control mask

ETH_DMATXNDESCRF_CPC_CRCPAD_INSERT

CRC Pad Control: CRC and Pad Insertion

ETH_DMATXNDESCRF_CPC_CRC_INSERT

CRC Pad Control: CRC Insertion (Disable Pad Insertion)

ETH_DMATXNDESCRF_CPC_DISABLE

CRC Pad Control: Disable CRC Insertion

ETH_DMATXNDESCRF_CPC_CRC_REPLACE

CRC Pad Control: CRC Replacement

ETH_DMATXNDESCRF_SAIC

SA Insertion Control mask

ETH_DMATXNDESCRF_SAIC_DISABLE

SA Insertion Control: Do not include the source address

ETH_DMATXNDESCRF_SAIC_INSERT

SA Insertion Control: Include or insert the source address

ETH_DMATXNDESCRF_SAIC_REPLACE

SA Insertion Control: Replace the source address

ETH_DMATXNDESCRF_THL

TCP Header Length

ETH_DMATXNDESCRF_TSE

TCP segmentation enable

ETH_DMATXNDESCRF_CIC

Checksum Insertion Control: 4 cases

ETH_DMATXNDESCRF_CIC_DISABLE

Do Nothing: Checksum Engine is disabled

ETH_DMATXNDESCRF_CIC_IPHDR_INSERT

Only IP header checksum calculation and insertion are enabled.

ETH_DMATXNDESCRF_CIC_IPHDR_PAYLOAD_INSERT

IP header checksum and payload checksum calculation and insertion are enabled, but pseudo header checksum is not calculated in hardware

ETH_DMATXNDESCRF_CIC_IPHDR_PAYLOAD_INSERT_PHDR_CALC

IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo header checksum is calculated in hardware.

ETH_DMATXNDESCRF_TPL

TCP Payload Length

ETH_DMATXNDESCRF_FL

Transmit End of Ring

ETH_DMATXNDESCWBF_TTS1

Buffer1 Address Pointer or TSO Header Address Pointer

ETH_DMATXNDESCWBF_TTS2

Buffer2 Address Pointer

ETH_DMATXNDESCWBF_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXNDESCWBF_CTXT

Context Type

ETH_DMATXNDESCWBF_FD

First Descriptor

ETH_DMATXNDESCWBF_LD

Last Descriptor

ETH_DMATXNDESCWBF_TTSS

Tx Timestamp Status

ETH_DMATXNDESCWBF_DP

Disable Padding

ETH_DMATXNDESCWBF_TTSE

Transmit Timestamp Enable

ETH_DMATXNDESCWBF_ES

Error summary: OR of the following bits: IHE || UF || ED || EC || LCO || PCE || NC || LCA || FF || JT

ETH_DMATXNDESCWBF_JT

Jabber Timeout

ETH_DMATXNDESCWBF_FF

Packet Flushed: DMA/MTL flushed the packet due to SW flush

ETH_DMATXNDESCWBF_PCE

Payload Checksum Error

ETH_DMATXNDESCWBF_LCA

Loss of Carrier: carrier lost during transmission

ETH_DMATXNDESCWBF_NC

No Carrier: no carrier signal from the transceiver

ETH_DMATXNDESCWBF_LCO

Late Collision: transmission aborted due to collision

ETH_DMATXNDESCWBF_EC

Excessive Collision: transmission aborted after 16 collisions

ETH_DMATXNDESCWBF_CC

Collision Count

ETH_DMATXNDESCWBF_ED

Excessive Deferral

ETH_DMATXNDESCWBF_UF

Underflow Error: late data arrival from the memory

ETH_DMATXNDESCWBF_DB

Deferred Bit

ETH_DMATXNDESCWBF_IHE

IP Header Error

ETH_DMATXCDESC_TTSL

Transmit Packet Timestamp Low

ETH_DMATXCDESC_TTSH

Transmit Packet Timestamp High

ETH_DMATXCDESC_IVT

Inner VLAN Tag

ETH_DMATXCDESC_MSS

Maximum Segment Size

ETH_DMATXCDESC_OWN

OWN bit: descriptor is owned by DMA engine

ETH_DMATXCDESC_CTXT

Context Type

ETH_DMATXCDESC_OSTC

One-Step Timestamp Correction Enable

ETH_DMATXCDESC_TCMSS

One-Step Timestamp Correction Input or MSS Valid

ETH_DMATXCDESC_CDE

Context Descriptor Error

ETH_DMATXCDESC_IVTIR

Inner VLAN Tag Insert or Replace Mask

ETH_DMATXCDESC_IVTIR_DISABLE

Do not add the inner VLAN tag.

ETH_DMATXCDESC_IVTIR_REMOVE

Remove the inner VLAN tag from the packets before transmission.

ETH_DMATXCDESC_IVTIR_INSERT

Insert the inner VLAN tag.

ETH_DMATXCDESC_IVTIR_REPLACE

Replace the inner VLAN tag.

ETH_DMATXCDESC_IVLTV

Inner VLAN Tag Valid

ETH_DMATXCDESC_VLTV

VLAN Tag Valid

ETH_DMATXCDESC_VT

VLAN Tag

ETH Duplex Mode**ETH_FULLDUPLEX_MODE*****ETH Error Code*****HAL_ETH_ERROR_NONE**

No error

HAL_ETH_ERROR_PARAM

Busy error

HAL_ETH_ERROR_BUSY

Parameter error

HAL_ETH_ERROR_TIMEOUT

Timeout error

HAL_ETH_ERROR_DMA

DMA transfer error

HAL_ETH_ERROR_MAC

MAC transfer error

ETH Exported Macros**__HAL_ETH_RESET_HANDLE_STATE****Description:**

- Reset ETH handle state.

Parameters:

- __HANDLE__: specifies the ETH handle.

Return value:

- None

__HAL_ETH_DMA_ENABLE_IT**Description:**

- Enables the specified ETHERNET DMA interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET DMA interrupt sources to be enabled

Return value:

- None

__HAL_ETH_DMA_DISABLE_IT**Description:**

- Disables the specified ETHERNET DMA interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET DMA interrupt sources to be disabled.

Return value:

- None

__HAL_ETH_DMA_GET_IT_SOURCE**Description:**

- Gets the ETHERNET DMA IT source enabled or disabled.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: ETH DMA IT Source enabled or disabled

[__HAL_ETH_DMA_GET_IT](#)**Description:**

- Gets the ETHERNET DMA IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt source to get .

Return value:

- The: state of ETH DMA IT (SET or RESET)

[__HAL_ETH_DMA_CLEAR_IT](#)**Description:**

- Clears the ETHERNET DMA IT pending bit.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the interrupt pending bit to clear.

Return value:

- None

[__HAL_ETH_DMA_GET_FLAG](#)**Description:**

- Checks whether the specified ETHERNET DMA flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag to check.

Return value:

- The: state of ETH DMA FLAG (SET or RESET).

[__HAL_ETH_DMA_CLEAR_FLAG](#)**Description:**

- Clears the specified ETHERNET DMA flag.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag to check.

Return value:

- The: state of ETH DMA FLAG (SET or RESET).

[__HAL_ETH_MAC_ENABLE_IT](#)**Description:**

- Enables the specified ETHERNET MAC interrupts.

Parameters:

- __HANDLE__: : ETH Handle
- __INTERRUPT__: specifies the ETHERNET MAC interrupt sources to be enabled

Return value:

- None

_HAL_ETH_MAC_DISABLE_IT

Description:

- Disables the specified ETHERNET MAC interrupts.

Parameters:

- _HANDLE_: : ETH Handle
- _INTERRUPT_: specifies the ETHERNET MAC interrupt sources to be enabled

Return value:

- None

_HAL_ETH_MAC_GET_IT

Description:

- Checks whether the specified ETHERNET MAC flag is set or not.

Parameters:

- _HANDLE_: ETH Handle
- _INTERRUPT_: specifies the flag to check.

Return value:

- The: state of ETH MAC IT (SET or RESET). External interrupt line 86 Connected to the ETH wakeup EXTI Line

ETH_WAKEUP_EXTI_LINE

_HAL_ETH_WAKEUP_EXTI_ENABLE_IT

Description:

- Enable the ETH WAKEUP Exti Line.

Parameters:

- _EXTI_LINE_: specifies the ETH WAKEUP Exti sources to be enabled.
 - ETH_WAKEUP_EXTI_LINE

Return value:

- None.

_HAL_ETH_WAKEUP_EXTI_GET_FLAG

Description:

- checks whether the specified ETH WAKEUP Exti interrupt flag is set or not.

Parameters:

- _EXTI_LINE_: specifies the ETH WAKEUP Exti sources to be cleared.
 - ETH_WAKEUP_EXTI_LINE

Return value:

- EXTI: ETH WAKEUP Line Status.

_HAL_ETH_WAKEUP_EXTI_CLEAR_FLAG

Description:

- Clear the ETH WAKEUP Exti flag.

Parameters:

- _EXTI_LINE_: specifies the ETH WAKEUP Exti sources to be cleared.

- ETH_WAKEUP_EXTI_LINE

Return value:

- None.

[__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_EDGE](#)**Description:**

- enable rising edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
 - ETH_WAKEUP_EXTI_LINE

Return value:

- None

[__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLING_EDGE](#)**Description:**

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
 - ETH_WAKEUP_EXTI_LINE

Return value:

- None

[__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE](#)**Description:**

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
 - ETH_WAKEUP_EXTI_LINE

Return value:

- None

[__HAL_ETH_WAKEUP_EXTI_GENERATE_SWIT](#)**Description:**

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled.
 - ETH_WAKEUP_EXTI_LINE

Return value:

- None

ETH frame settings[ETH_MAX_PACKET_SIZE](#)

ETH_HEADER + 2*VLAN_TAG + MAX_ETH_PAYLOAD + ETH_CRC

[ETH_HEADER](#)

6 byte Dest addr, 6 byte Src addr, 2 byte length/type

ETH_CRC

Ethernet CRC

ETH_VLAN_TAG

optional 802.1q VLAN Tag

ETH_MIN_PAYLOAD

Minimum Ethernet payload size

ETH_MAX_PAYLOAD

Maximum Ethernet payload size

ETH_JUMBO_FRAME_PAYLOAD

Jumbo frame payload size

ETH Inter Packet Gap**ETH_INTERPACKETGAP_96BIT****ETH_INTERPACKETGAP_88BIT****ETH_INTERPACKETGAP_80BIT****ETH_INTERPACKETGAP_72BIT****ETH_INTERPACKETGAP_64BIT****ETH_INTERPACKETGAP_56BIT****ETH_INTERPACKETGAP_48BIT****ETH_INTERPACKETGAP_40BIT*****ETH MAC addresses*****ETH_MAC_ADDRESS0****ETH_MAC_ADDRESS1****ETH_MAC_ADDRESS2****ETH_MAC_ADDRESS3*****ETH MAC Interrupts*****ETH_MAC_RX_STATUS_IT****ETH_MAC_TX_STATUS_IT****ETH_MAC_TIMESTAMP_IT****ETH_MAC_LPI_IT****ETH_MAC_PMT_IT****ETH_MAC_PHY_IT*****ETH MAC Rx Tx Status***

ETH_RECEIVE_WATCHDOG_TIMEOUT

ETH_EXCESSIVE_COLLISIONS

ETH_LATE_COLLISIONS

ETH_EXCESSIVE_DEFERRAL

ETH LOSS_OF_CARRIER

ETH_NO_CARRIER

ETH_TRANSMIT_JABBR_TIMEOUT

ETH MAC Wake Up Event

ETH_WAKEUP_PACKET_RECIEVED

ETH_MAGIC_PACKET_RECIEVED

ETH Pause Low Threshold

ETH_PAUSELOWTHRESHOLD_MINUS_4

ETH_PAUSELOWTHRESHOLD_MINUS_28

ETH_PAUSELOWTHRESHOLD_MINUS_36

ETH_PAUSELOWTHRESHOLD_MINUS_144

ETH_PAUSELOWTHRESHOLD_MINUS_256

ETH_PAUSELOWTHRESHOLD_MINUS_512

ETH Preamble Length

ETH_PREAMBLELENGTH_7

ETH_PREAMBLELENGTH_5

ETH_PREAMBLELENGTH_3

ETH Receive Mode

ETH_RECEIVESTOREFORWARD

ETH_RECEIVETHRESHOLD8_64

ETH_RECEIVETHRESHOLD8_32

ETH_RECEIVETHRESHOLD8_96

ETH_RECEIVETHRESHOLD8_128

ETH Rx Checksum Status

ETH_CHECKSUM_BYPASSED

ETH_CHECKSUM_IP_HEADER_ERROR

ETH_CHECKSUM_IP_PAYLOAD_ERROR

ETH Rx DMA Burst Length

ETH_RXDMABURSTLENGTH_1BEAT

ETH_RXDMABURSTLENGTH_2BEAT

ETH_RXDMABURSTLENGTH_4BEAT

ETH_RXDMABURSTLENGTH_8BEAT

ETH_RXDMABURSTLENGTH_16BEAT

ETH_RXDMABURSTLENGTH_32BEAT

ETH Rx Error Code

ETH_DRIBBLE_BIT_ERROR

ETH_RECEIVE_ERROR

ETH_RECEIVE_OVERFLOW

ETH_WATCHDOG_TIMEOUT

ETH_GIANT_PACKET

ETH_CRC_ERROR

ETH Rx IP Header Type

ETH_IP_HEADER_IPV4

ETH_IP_HEADER_IPV6

ETH Rx L3 Filter Status

ETH_L3_FILTER0_MATCH

ETH_L3_FILTER1_MATCH

ETH Rx L4 Filter Status

ETH_L4_FILTER0_MATCH

ETH_L4_FILTER1_MATCH

ETH Rx MAC Filter Status

ETH_HASH_FILTER_PASS

ETH_VLAN_FILTER_PASS

ETH_DEST_ADDRESS_FAIL

ETH_SOURCE_ADDRESS_FAIL

ETH Rx Payload Type

ETH_IP_PAYLOAD_UNKNOWN

ETH_IP_PAYLOAD_UDP

ETH_IP_PAYLOAD_TCP

ETH_IP_PAYLOAD_ICMPN

ETH Source Addr Control

ETH_SOURCEADDRESS_DISABLE

ETH_SOURCEADDRESS_INSERT_ADDR0

ETH_SOURCEADDRESS_INSERT_ADDR1

ETH_SOURCEADDRESS_REPLACE_ADDR0

ETH_SOURCEADDRESS_REPLACE_ADDR1

ETH Speed

ETH_SPEED_10M

ETH_SPEED_100M

ETH Transmit Mode

ETH_TRANSMITSTOREFORWARD

ETH_TRANSMITTHRESHOLD_32

ETH_TRANSMITTHRESHOLD_64

ETH_TRANSMITTHRESHOLD_96

ETH_TRANSMITTHRESHOLD_128

ETH_TRANSMITTHRESHOLD_192

ETH_TRANSMITTHRESHOLD_256

ETH_TRANSMITTHRESHOLD_384

ETH_TRANSMITTHRESHOLD_512

ETH Tx DMA Burst Length

ETH_TXDMABURSTLENGTH_1BEAT

ETH_TXDMABURSTLENGTH_2BEAT

ETH_TXDMABURSTLENGTH_4BEAT

ETH_TXDMABURSTLENGTH_8BEAT

ETH_TXDMABURSTLENGTH_16BEAT

ETH_TXDMABURSTLENGTH_32BEAT

ETH Tx Packet Attributes

ETH_TX_PACKETS_FEATURES_CSUM
ETH_TX_PACKETS_FEATURES_SAIC
ETH_TX_PACKETS_FEATURES_VLANTAG
ETH_TX_PACKETS_FEATURES_INNERTVLANTAG
ETH_TX_PACKETS_FEATURES_TSO
ETH_TX_PACKETS_FEATURES_CRCPAD
ETH Tx Packet Checksum Control
ETH_CHECKSUM_DISABLE
ETH_CHECKSUM_IPHDR_INSERT
ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT
ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT_PHDR_CALC
ETH Tx Packet CRC Pad Control
ETH_CRC_PAD_DISABLE
ETH_CRC_PAD_INSERT
ETH_CRC_INSERT
ETH_CRC_REPLACE
ETH Tx Packet Inner VLAN Control
ETH_INNER_VLAN_DISABLE
ETH_INNER_VLAN_REMOVE
ETH_INNER_VLAN_INSERT
ETH_INNER_VLAN_REPLACE
ETH Tx Packet Source Addr Control
ETH_SRC_ADDR_CONTROL_DISABLE
ETH_SRC_ADDR_INSERT
ETH_SRC_ADDR_REPLACE
ETH Tx Packet VLAN Control
ETH_VLAN_DISABLE
ETH_VLAN_REMOVE
ETH_VLAN_INSERT
ETH_VLAN_REPLACE

ETH VLAN Tag Comparison`ETH_VLANTAGCOMPARISON_16BIT``ETH_VLANTAGCOMPARISON_12BIT`*ETH Watchdog Timeout*`ETH_WATCHDOGTIMEOUT_2KB``ETH_WATCHDOGTIMEOUT_3KB``ETH_WATCHDOGTIMEOUT_4KB``ETH_WATCHDOGTIMEOUT_5KB``ETH_WATCHDOGTIMEOUT_6KB``ETH_WATCHDOGTIMEOUT_7KB``ETH_WATCHDOGTIMEOUT_8KB``ETH_WATCHDOGTIMEOUT_9KB``ETH_WATCHDOGTIMEOUT_10KB``ETH_WATCHDOGTIMEOUT_11KB``ETH_WATCHDOGTIMEOUT_12KB``ETH_WATCHDOGTIMEOUT_13KB``ETH_WATCHDOGTIMEOUT_14KB``ETH_WATCHDOGTIMEOUT_15KB``ETH_WATCHDOGTIMEOUT_16KB`

22 HAL ETH Extension Driver

22.1 ETHE Ex Firmware driver registers structures

22.1.1 ETH_RxVLANConfigTypeDef

Data Fields

- *FunctionalState InnerVLANTagInStatus*
- *uint32_t StripInnerVLANTag*
- *FunctionalState InnerVLANTag*
- *FunctionalState DoubleVLANProcessing*
- *FunctionalState VLANTagHashTableMatch*
- *FunctionalState VLANTagInStatus*
- *uint32_t StripVLANTag*
- *uint32_t VLANTypeCheck*
- *FunctionalState VLANTagInverseMatch*

Field Documentation

- ***FunctionalState ETH_RxVLANConfigTypeDef::InnerVLANTagInStatus***
Enables or disables Inner VLAN Tag in Rx Status
- ***uint32_t ETH_RxVLANConfigTypeDef::StripInnerVLANTag***
Sets the Inner VLAN Tag Stripping on Receive This parameter can be a value of **ETHE Ex Rx Inner VLAN Tag Stripping**
- ***FunctionalState ETH_RxVLANConfigTypeDef::InnerVLANTag***
Enables or disables Inner VLAN Tag
- ***FunctionalState ETH_RxVLANConfigTypeDef::DoubleVLANProcessing***
Enable or Disable double VLAN processing
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagHashTableMatch***
Enable or Disable VLAN Tag Hash Table Match
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagInStatus***
Enable or Disable VLAN Tag in Rx status
- ***uint32_t ETH_RxVLANConfigTypeDef::StripVLANTag***
Set the VLAN Tag Stripping on Receive This parameter can be a value of **ETHE Ex Rx VLAN Tag Stripping**
- ***uint32_t ETH_RxVLANConfigTypeDef::VLANTypeCheck***
Enable or Disable VLAN Type Check This parameter can be a value of **ETHE Ex VLAN Type Check**
- ***FunctionalState ETH_RxVLANConfigTypeDef::VLANTagInverseMatch***
Enable or disable VLAN Tag Inverse Match

22.1.2 ETH_TxVLANConfigTypeDef

Data Fields

- *FunctionalState SourceTxDesc*
- *FunctionalState SVLANType*
- *uint32_t VLANTagControl*

Field Documentation

- ***FunctionalState ETH_TxVLANConfigTypeDef::SourceTxDesc***
Enable or Disable VLAN tag source from DMA tx descriptors
- ***FunctionalState ETH_TxVLANConfigTypeDef::SVLANType***
Enable or Disable insertion of SVLAN type
- ***uint32_t ETH_TxVLANConfigTypeDef::VLANTagControl***
Sets the VLAN tag control in tx packets This parameter can be a value of ***ETHEx_VLAN_Tag_Control***

22.1.3 ETH_L3FilterConfigTypeDef

Data Fields

- ***uint32_t Protocol***
- ***uint32_t SrcAddrFilterMatch***
- ***uint32_t DestAddrFilterMatch***
- ***uint32_t SrcAddrHigherBitsMatch***
- ***uint32_t DestAddrHigherBitsMatch***
- ***uint32_t Ip4SrcAddr***
- ***uint32_t Ip4DestAddr***
- ***uint32_t Ip6Addr***

Field Documentation

- ***uint32_t ETH_L3FilterConfigTypeDef::Protocol***
Sets the L3 filter protocol to IPv4 or IPv6 This parameter can be a value of ***ETHEx_L3_Protocol***
- ***uint32_t ETH_L3FilterConfigTypeDef::SrcAddrFilterMatch***
Sets the L3 filter source address match This parameter can be a value of ***ETHEx_L3_Source_Match***
- ***uint32_t ETH_L3FilterConfigTypeDef::DestAddrFilterMatch***
Sets the L3 filter destination address match This parameter can be a value of ***ETHEx_L3_Destination_Match***
- ***uint32_t ETH_L3FilterConfigTypeDef::SrcAddrHigherBitsMatch***
Sets the L3 filter source address higher bits match This parameter can be a value from 0 to 31
- ***uint32_t ETH_L3FilterConfigTypeDef::DestAddrHigherBitsMatch***
Sets the L3 filter destination address higher bits match This parameter can be a value from 0 to 31
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip4SrcAddr***
Sets the L3 filter IPv4 source address if IPv4 protocol is used This parameter can be a value from 0x0 to 0xFFFFFFFF
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip4DestAddr***
Sets the L3 filter IPv4 destination address if IPv4 protocol is used This parameter can be a value from 0 to 0xFFFFFFFF
- ***uint32_t ETH_L3FilterConfigTypeDef::Ip6Addr[4]***
Sets the L3 filter IPv6 address if IPv6 protocol is used This parameter must be a table of 4 words (4* 32 bits)

22.1.4 ETH_L4FilterConfigTypeDef

Data Fields

- ***uint32_t Protocol***
- ***uint32_t SrcPortFilterMatch***
- ***uint32_t DestPortFilterMatch***

- `uint32_t SourcePort`
- `uint32_t DestinationPort`

Field Documentation

- `uint32_t ETH_L4FilterConfigTypeDef::Protocol`
Sets the L4 filter protocol to TCP or UDP This parameter can be a value of **ETHEEx L4 Protocol**
- `uint32_t ETH_L4FilterConfigTypeDef::SrcPortFilterMatch`
Sets the L4 filter source port match This parameter can be a value of **ETHEEx L4 Source Match**
- `uint32_t ETH_L4FilterConfigTypeDef::DestPortFilterMatch`
Sets the L4 filter destination port match This parameter can be a value of **ETHEEx L4 Destination Match**
- `uint32_t ETH_L4FilterConfigTypeDef::SourcePort`
Sets the L4 filter source port This parameter must be a value from 0x0 to 0xFFFF
- `uint32_t ETH_L4FilterConfigTypeDef::DestinationPort`
Sets the L4 filter destination port This parameter must be a value from 0x0 to 0xFFFF

22.2 ETHEEx Firmware driver API description

22.2.1 Extended features functions

This section provides functions allowing to:

- Configure ARP offload module
- Configure L3 and L4 filters
- Configure Extended VLAN features
- Configure Energy Efficient Ethernet module

This section contains the following APIs:

- [`HAL_ETHEEx_EnableARPOffload`](#)
- [`HAL_ETHEEx_DisableARPOffload`](#)
- [`HAL_ETHEEx_SetARPAddressMatch`](#)
- [`HAL_ETHEEx_SetL4FilterConfig`](#)
- [`HAL_ETHEEx_GetL4FilterConfig`](#)
- [`HAL_ETHEEx_SetL3FilterConfig`](#)
- [`HAL_ETHEEx_GetL3FilterConfig`](#)
- [`HAL_ETHEEx_EnableL3L4Filtering`](#)
- [`HAL_ETHEEx_DisableL3L4Filtering`](#)
- [`HAL_ETHEEx_GetRxVLANConfig`](#)
- [`HAL_ETHEEx_SetRxVLANConfig`](#)
- [`HAL_ETHEEx_SetVLANHashTable`](#)
- [`HAL_ETHEEx_GetTxVLANConfig`](#)
- [`HAL_ETHEEx_SetTxVLANConfig`](#)
- [`HAL_ETHEEx_SetTxVLANIdentifier`](#)
- [`HAL_ETHEEx_EnableVLANProcessing`](#)
- [`HAL_ETHEEx_DisableVLANProcessing`](#)
- [`HAL_ETHEEx_EnterLPIMode`](#)
- [`HAL_ETHEEx_ExitLPIMode`](#)
- [`HAL_ETHEEx_GetMACLPIEvent`](#)

22.2.2 Detailed description of functions

HAL_ETHEX_EnableARPOffload

Function name

```
void HAL_ETHEX_EnableARPOffload (ETH_HandleTypeDef * heth)
```

Function description

Enables ARP Offload.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEX_DisableARPOffload

Function name

```
void HAL_ETHEX_DisableARPOffload (ETH_HandleTypeDef * heth)
```

Function description

Disables ARP Offload.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEX_SetARPAddressMatch

Function name

```
void HAL_ETHEX_SetARPAddressMatch (ETH_HandleTypeDef * heth, uint32_t ipAddress)
```

Function description

Set the ARP Match IP address.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **ipAddress:** IP Address to be matched for incoming ARP requests

Return values

- **None:**

HAL_ETHEX_EnableL3L4Filtering

Function name

```
void HAL_ETHEX_EnableL3L4Filtering (ETH_HandleTypeDef * heth)
```

Function description

Enables L3 and L4 filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEX_DisableL3L4Filtering

Function name

```
void HAL_ETHEX_DisableL3L4Filtering (ETH_HandleTypeDef * heth)
```

Function description

Disables L3 and L4 filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEX_GetL3FilterConfig

Function name

```
HAL_StatusTypeDef HAL_ETHEX_GetL3FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter,  
ETH_L3FilterConfigTypeDef * pL3FilterConfig)
```

Function description

Configures the L3 Filter, this function allow to: set the layer 3 protocol to be matched (IPv4 or IPv6) enable/disable L3 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L3 filter to configured, this parameter must be one of the following ETH_L3_FILTER_0 ETH_L3_FILTER_1
- **pL3FilterConfig:** pointer to a ETH_L3FilterConfigTypeDef structure that will contain the L3 filter configuration.

Return values

- **HAL:** status

HAL_ETHEX_GetL4FilterConfig

Function name

```
HAL_StatusTypeDef HAL_ETHEX_GetL4FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter,  
ETH_L4FilterConfigTypeDef * pL4FilterConfig)
```

Function description

Configures the L4 Filter, this function allow to: set the layer 4 protocol to be matched (TCP or UDP) enable/disable L4 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

- **Filter:** L4 filter to configured, this parameter must be one of the following ETH_L4_FILTER_0
ETH_L4_FILTER_1
- **pL4FilterConfig:** pointer to a ETH_L4FilterConfigTypeDef structure that contains L4 filter configuration.

Return values

- **HAL:** status

HAL_ETHEX_SetL3FilterConfig

Function name

```
HAL_StatusTypeDef HAL_ETHEX_SetL3FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter,  
ETH_L3FilterConfigTypeDef * pL3FilterConfig)
```

Function description

Configures the L3 Filter, this function allow to: set the layer 3 protocol to be matched (IPv4 or IPv6) enable/disable L3 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L3 filter to configured, this parameter must be one of the following ETH_L3_FILTER_0
ETH_L3_FILTER_1
- **pL3FilterConfig:** pointer to a ETH_L3FilterConfigTypeDef structure that contains L3 filter configuration.

Return values

- **HAL:** status

HAL_ETHEX_SetL4FilterConfig

Function name

```
HAL_StatusTypeDef HAL_ETHEX_SetL4FilterConfig (ETH_HandleTypeDef * heth, uint32_t Filter,  
ETH_L4FilterConfigTypeDef * pL4FilterConfig)
```

Function description

Configures the L4 Filter, this function allow to: set the layer 4 protocol to be matched (TCP or UDP) enable/disable L4 source/destination port perfect/inverse match.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **Filter:** L4 filter to configured, this parameter must be one of the following ETH_L4_FILTER_0
ETH_L4_FILTER_1
- **pL4FilterConfig:** pointer to a ETH_L4FilterConfigTypeDef structure that contains L4 filter configuration.

Return values

- **HAL:** status

HAL_ETHEX_EnableVLANProcessing

Function name

```
void HAL_ETHEX_EnableVLANProcessing (ETH_HandleTypeDef * heth)
```

Function description

Enables the VLAN Tag Filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEX_DisableVLANProcessing

Function name

void HAL_ETHEX_DisableVLANProcessing (ETH_HandleTypeDef * heth)

Function description

Disables the VLAN Tag Filtering process.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None.:**

HAL_ETHEX_GetRxVLANConfig

Function name

**HAL_StatusTypeDef HAL_ETHEX_GetRxVLANConfig (ETH_HandleTypeDef * heth,
ETH_RxVLANConfigTypeDef * pVlanConfig)**

Function description

Get the VLAN Configuration for Receive Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pVlanConfig:** pointer to a ETH_RxVLANConfigTypeDef structure that will contain the VLAN filter configuration.

Return values

- **HAL:** status

HAL_ETHEX_SetRxVLANConfig

Function name

**HAL_StatusTypeDef HAL_ETHEX_SetRxVLANConfig (ETH_HandleTypeDef * heth,
ETH_RxVLANConfigTypeDef * pVlanConfig)**

Function description

Set the VLAN Configuration for Receive Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **pVlanConfig:** pointer to a ETH_RxVLANConfigTypeDef structure that contains VLAN filter configuration.

Return values

- **HAL:** status

HAL_ETHEX_SetVLANHashTable

Function name

```
void HAL_ETHEX_SetVLANHashTable (ETH_HandleTypeDef * heth, uint32_t VLANHashTable)
```

Function description

Set the VLAN Hash Table.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANHashTable:** VLAN hash table 16 bit value

Return values

- **None:**

HAL_ETHEX_GetTxVLANConfig

Function name

```
HAL_StatusTypeDef HAL_ETHEX_GetTxVLANConfig (ETH_HandleTypeDef * heth, uint32_t VLANTag,  
ETH_TxVLANConfigTypeDef * pVlanConfig)
```

Function description

Get the VLAN Configuration for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **pVlanConfig:** pointer to a ETH_TxVLANConfigTypeDef structure that will contain the Tx VLAN filter configuration.

Return values

- **HAL:** Status.

HAL_ETHEX_SetTxVLANConfig

Function name

```
HAL_StatusTypeDef HAL_ETHEX_SetTxVLANConfig (ETH_HandleTypeDef * heth, uint32_t VLANTag,  
ETH_TxVLANConfigTypeDef * pVlanConfig)
```

Function description

Set the VLAN Configuration for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **pVlanConfig:** pointer to a ETH_TxVLANConfigTypeDef structure that contains Tx VLAN filter configuration.

Return values

- **HAL:** Status

HAL_ETHEX_SetTxVLANIdentifier

Function name

```
void HAL_ETHEX_SetTxVLANIdentifier (ETH_HandleTypeDef * heth, uint32_t VLANTag, uint32_t VLANIdentifier)
```

Function description

Set the VLAN Tag Identifier for Transmit Packets.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **VLANTag:** Selects the vlan tag, this parameter must be one of the following ETH_OUTER_TX_VLANTAG ETH_INNER_TX_VLANTAG
- **VLANIdentifier:** VLAN Identifier 16 bit value

Return values

- **None:**

HAL_ETHEX_EnterLPIMode

Function name

```
void HAL_ETHEX_EnterLPIMode (ETH_HandleTypeDef * heth, FunctionalState TxAutomate, FunctionalState TxClockStop)
```

Function description

Enters the Low Power Idle (LPI) mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **TxAutomate:** Enable/Disable automate enter/exit LPI mode.
- **TxClockStop:** Enable/Disable Tx clock stop in LPI mode.

Return values

- **None:**

HAL_ETHEX_ExitLPIMode

Function name

```
void HAL_ETHEX_ExitLPIMode (ETH_HandleTypeDef * heth)
```

Function description

Exits the Low Power Idle (LPI) mode.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **None:**

HAL_ETHEX_GetMACLPIEvent

Function name

```
uint32_t HAL_ETHEX_GetMACLPIEvent (ETH_HandleTypeDef * heth)
```

Function description

Returns the ETH MAC LPI event.

Parameters

- **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- **ETH:** MAC WakeUp event

22.3 ETHEEx Firmware driver defines

22.3.1 ETHEEx

ETHEEx L3 Destination Match

`ETH_L3_DEST_ADDR_PERFECT_MATCH_ENABLE`

`ETH_L3_DEST_ADDR_INVERSE_MATCH_ENABLE`

`ETH_L3_DEST_ADDR_MATCH_DISABLE`

ETHEEx L3 Filter

`ETH_L3_FILTER_0`

`ETH_L3_FILTER_1`

ETHEEx L3 Protocol

`ETH_L3_IPV6_MATCH`

`ETH_L3_IPV4_MATCH`

ETHEEx L3 Source Match

`ETH_L3_SRC_ADDR_PERFECT_MATCH_ENABLE`

`ETH_L3_SRC_ADDR_INVERSE_MATCH_ENABLE`

`ETH_L3_SRC_ADDR_MATCH_DISABLE`

ETHEEx L4 Destination Match

`ETH_L4_DEST_PORT_PERFECT_MATCH_ENABLE`

`ETH_L4_DEST_PORT_INVERSE_MATCH_ENABLE`

`ETH_L4_DEST_PORT_MATCH_DISABLE`

ETHEEx L4 Filter

`ETH_L4_FILTER_0`

`ETH_L4_FILTER_1`

ETHEEx L4 Protocol

`ETH_L4_UDP_MATCH`

ETH_L4_TCP_MATCH

ETHEx L4 Source Match

ETH_L4_SRC_PORT_PERFECT_MATCH_ENABLE

ETH_L4_SRC_PORT_INVERSE_MATCH_ENABLE

ETH_L4_SRC_PORT_MATCH_DISABLE

ETHEx LPI Event

ETH_TX_LPI_ENTRY

ETH_TX_LPI_EXIT

ETH_RX_LPI_ENTRY

ETH_RX_LPI_EXIT

ETHEx Rx Inner VLAN Tag Stripping

ETH_INNERVLANTAGRXSTRIPPING_NONE

ETH_INNERVLANTAGRXSTRIPPING_IFPASS

ETH_INNERVLANTAGRXSTRIPPING_IFFAILS

ETH_INNERVLANTAGRXSTRIPPING_ALWAYS

ETHEx Rx VLAN Tag Stripping

ETH_VLANTAGRXSTRIPPING_NONE

ETH_VLANTAGRXSTRIPPING_IFPASS

ETH_VLANTAGRXSTRIPPING_IFFAILS

ETH_VLANTAGRXSTRIPPING_ALWAYS

ETHEx Tx VLAN Tag

ETH_INNER_TX_VLANTAG

ETH_OUTER_TX_VLANTAG

ETHEx VLAN Tag Control

ETH_VLANTAGCONTROL_NONE

ETH_VLANTAGCONTROL_DELETE

ETH_VLANTAGCONTROL_INSERT

ETH_VLANTAGCONTROL_REPLACE

ETHEx VLAN Type Check

ETH_VLANTYPECHECK_DISABLE

ETH_VLANTYPECHECK_SVLAN

ETH_VLANTYPECHECK_CVLAN

23 HAL FDCAN Generic Driver

23.1 FDCAN Firmware driver registers structures

23.1.1 FDCAN_InitTypeDef

Data Fields

- *uint32_t FrameFormat*
- *uint32_t Mode*
- *FunctionalState AutoRetransmission*
- *FunctionalState TransmitPause*
- *FunctionalState ProtocolException*
- *uint32_t NominalPrescaler*
- *uint32_t NominalSyncJumpWidth*
- *uint32_t NominalTimeSeg1*
- *uint32_t NominalTimeSeg2*
- *uint32_t DataPrescaler*
- *uint32_t DataSyncJumpWidth*
- *uint32_t DataTimeSeg1*
- *uint32_t DataTimeSeg2*
- *uint32_t MessageRAMOffset*
- *uint32_t StdFiltersNbr*
- *uint32_t ExtFiltersNbr*
- *uint32_t RxFifo0ElmtsNbr*
- *uint32_t RxFifo0ElmtSize*
- *uint32_t RxFifo1ElmtsNbr*
- *uint32_t RxFifo1ElmtSize*
- *uint32_t RxBuffersNbr*
- *uint32_t RxBufferSize*
- *uint32_t TxEventsNbr*
- *uint32_t TxBuffersNbr*
- *uint32_t TxFifoQueueElmtsNbr*
- *uint32_t TxFifoQueueMode*
- *uint32_t TxElmtSize*

Field Documentation

- ***uint32_t FDCAN_InitTypeDef::FrameFormat***
Specifies the FDCAN frame format. This parameter can be a value of **FDCAN Frame Format**
- ***uint32_t FDCAN_InitTypeDef::Mode***
Specifies the FDCAN mode. This parameter can be a value of **FDCAN Operating Mode**
- ***FunctionalState FDCAN_InitTypeDef::AutoRetransmission***
Enable or disable the automatic retransmission mode. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::TransmitPause***
Enable or disable the Transmit Pause feature. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::ProtocolException***

Enable or disable the Protocol Exception Handling. This parameter can be set to ENABLE or DISABLE

- **`uint32_t FDCAN_InitTypeDef::NominalPrescaler`**

Specifies the value by which the oscillator frequency is divided for generating the nominal bit time quanta. This parameter must be a number between 1 and 512

- **`uint32_t FDCAN_InitTypeDef::NominalSyncJumpWidth`**

Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter must be a number between 1 and 128

- **`uint32_t FDCAN_InitTypeDef::NominalTimeSeg1`**

Specifies the number of time quanta in Bit Segment 1. This parameter must be a number between 2 and 256

- **`uint32_t FDCAN_InitTypeDef::NominalTimeSeg2`**

Specifies the number of time quanta in Bit Segment 2. This parameter must be a number between 2 and 128

- **`uint32_t FDCAN_InitTypeDef::DataPrescaler`**

Specifies the value by which the oscillator frequency is divided for generating the data bit time quanta. This parameter must be a number between 1 and 32

- **`uint32_t FDCAN_InitTypeDef::DataSyncJumpWidth`**

Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a data bit to perform resynchronization. This parameter must be a number between 1 and 16

- **`uint32_t FDCAN_InitTypeDef::DataTimeSeg1`**

Specifies the number of time quanta in Data Bit Segment 1. This parameter must be a number between 1 and 32

- **`uint32_t FDCAN_InitTypeDef::DataTimeSeg2`**

Specifies the number of time quanta in Data Bit Segment 2. This parameter must be a number between 1 and 16

- **`uint32_t FDCAN_InitTypeDef::MessageRAMOffset`**

Specifies the message RAM start address. This parameter must be a number between 0 and 2560

- **`uint32_t FDCAN_InitTypeDef::StdFiltersNbr`**

Specifies the number of standard Message ID filters. This parameter must be a number between 0 and 128

- **`uint32_t FDCAN_InitTypeDef::ExtFiltersNbr`**

Specifies the number of extended Message ID filters. This parameter must be a number between 0 and 64

- **`uint32_t FDCAN_InitTypeDef::RxFifo0ElmtsNbr`**

Specifies the number of Rx FIFO 0 Elements. This parameter must be a number between 0 and 64

- **`uint32_t FDCAN_InitTypeDef::RxFifo0ElmtSize`**

Specifies the Data Field Size in an Rx FIFO 0 element. This parameter can be a value of **FDCAN Data Field Size**

- **`uint32_t FDCAN_InitTypeDef::RxFifo1ElmtsNbr`**

Specifies the number of Rx FIFO 1 Elements. This parameter must be a number between 0 and 64

- **`uint32_t FDCAN_InitTypeDef::RxFifo1ElmtSize`**

Specifies the Data Field Size in an Rx FIFO 1 element. This parameter can be a value of **FDCAN Data Field Size**

- **`uint32_t FDCAN_InitTypeDef::RxBuffersNbr`**

Specifies the number of Dedicated Rx Buffer elements. This parameter must be a number between 0 and 64

- **`uint32_t FDCAN_InitTypeDef::RxBufferSize`**

Specifies the Data Field Size in an Rx Buffer element. This parameter can be a value of **FDCAN Data Field Size**

- **`uint32_t FDCAN_InitTypeDef::TxEventsNbr`**
Specifies the number of Tx Event FIFO elements. This parameter must be a number between 0 and 32
- **`uint32_t FDCAN_InitTypeDef::TxBuffersNbr`**
Specifies the number of Dedicated Tx Buffers. This parameter must be a number between 0 and 32
- **`uint32_t FDCAN_InitTypeDef::TxFifoQueueElmtsNbr`**
Specifies the number of Tx Buffers used for Tx FIFO/Queue. This parameter must be a number between 0 and 32
- **`uint32_t FDCAN_InitTypeDef::TxFifoQueueMode`**
Tx FIFO/Queue Mode selection. This parameter can be a value of **FDCAN Tx FIFO/Queue Mode**
- **`uint32_t FDCAN_InitTypeDef::TxElmtSize`**
Specifies the Data Field Size in a Tx Element. This parameter can be a value of **FDCAN Data Field Size**

23.1.2 FDCAN_ClkCalUnitTypeDef

Data Fields

- **`uint32_t ClockCalibration`**
- **`uint32_t ClockDivider`**
- **`uint32_t MinOscClkPeriods`**
- **`uint32_t CalFieldLength`**
- **`uint32_t TimeQuantaPerBitTime`**
- **`uint32_t WatchdogStartValue`**

Field Documentation

- **`uint32_t FDCAN_ClkCalUnitTypeDef::ClockCalibration`**
Enable or disable the clock calibration. This parameter can be set to ENABLE or DISABLE
- **`uint32_t FDCAN_ClkCalUnitTypeDef::ClockDivider`**
Specifies the FDCAN kernel clock divider when the clock calibration is bypassed. This parameter can be a value of **FDCAN Clock Divider**
- **`uint32_t FDCAN_ClkCalUnitTypeDef::MinOscClkPeriods`**
Configures the minimum number of periods in two CAN bit times. The actual configured number of periods is MinOscClkPeriods x 32. This parameter must be a number between 0x00 and 0xFF
- **`uint32_t FDCAN_ClkCalUnitTypeDef::CalFieldLength`**
Specifies the calibration field length. This parameter can be a value of **FDCAN Calibration Field Length**
- **`uint32_t FDCAN_ClkCalUnitTypeDef::TimeQuantaPerBitTime`**
Configures the number of time quanta per bit time. This parameter must be a number between 4 and 25
- **`uint32_t FDCAN_ClkCalUnitTypeDef::WatchdogStartValue`**
Start value of the Calibration Watchdog Counter. If set to zero the counter is disabled. This parameter must be a number between 0x0000 and 0xFFFF

23.1.3 FDCAN_FilterTypeDef

Data Fields

- **`uint32_t IdType`**
- **`uint32_t FilterIndex`**
- **`uint32_t FilterType`**
- **`uint32_t FilterConfig`**
- **`uint32_t FilterID1`**

- `uint32_t FilterID2`
- `uint32_t RxBufferIndex`
- `uint32_t IsCalibrationMsg`

Field Documentation

- `uint32_t FDCAN_FilterTypeDef::IdType`

Specifies the identifier type. This parameter can be a value of **FDCAN ID Type**

- `uint32_t FDCAN_FilterTypeDef::FilterIndex`

Specifies the filter which will be initialized. This parameter must be a number between:

- 0 and 127, if IdType is FDCAN_STANDARD_ID
- 0 and 63, if IdType is FDCAN_EXTENDED_ID

- `uint32_t FDCAN_FilterTypeDef::FilterType`

Specifies the filter type. This parameter can be a value of **FDCAN Filter Type**. The value FDCAN_EXT_FILTER_RANGE_NO_EIDM is permitted only when IdType is FDCAN_EXTENDED_ID. This parameter is ignored if FilterConfig is set to FDCAN_FILTER_TO_RXBUFFER

- `uint32_t FDCAN_FilterTypeDef::FilterConfig`

Specifies the filter configuration. This parameter can be a value of **FDCAN Filter Configuration**

- `uint32_t FDCAN_FilterTypeDef::FilterID1`

Specifies the filter identification 1. This parameter must be a number between:

- 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
- 0 and 0xFFFFFFFF, if IdType is FDCAN_EXTENDED_ID

- `uint32_t FDCAN_FilterTypeDef::FilterID2`

Specifies the filter identification 2. This parameter is ignored if FilterConfig is set to FDCAN_FILTER_TO_RXBUFFER. This parameter must be a number between:

- 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
- 0 and 0xFFFFFFFF, if IdType is FDCAN_EXTENDED_ID

- `uint32_t FDCAN_FilterTypeDef::RxBufferIndex`

Contains the index of the Rx buffer in which the matching message will be stored. This parameter must be a number between 0 and 63. This parameter is ignored if FilterConfig is different from FDCAN_FILTER_TO_RXBUFFER

- `uint32_t FDCAN_FilterTypeDef::IsCalibrationMsg`

Specifies whether the filter is configured for calibration messages. This parameter is ignored if FilterConfig is different from FDCAN_FILTER_TO_RXBUFFER. This parameter can be:

- 0 : ordinary message
- 1 : calibration message

23.1.4 FDCAN_TxHeaderTypeDef

Data Fields

- `uint32_t Identifier`
- `uint32_t IdType`
- `uint32_t TxFrameType`
- `uint32_t DataLength`
- `uint32_t ErrorStateIndicator`
- `uint32_t BitRateSwitch`
- `uint32_t FDFormat`
- `uint32_t TxEventFifoControl`
- `uint32_t MessageMarker`

Field Documentation

- **`uint32_t FDCAN_TxHeaderTypeDef::Identifier`**
Specifies the identifier. This parameter must be a number between:
 - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
 - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- **`uint32_t FDCAN_TxHeaderTypeDef::IdType`**
Specifies the identifier type for the message that will be transmitted. This parameter can be a value of **FDCAN ID Type**
- **`uint32_t FDCAN_TxHeaderTypeDef::TxFrameType`**
Specifies the frame type of the message that will be transmitted. This parameter can be a value of **FDCAN Frame Type**
- **`uint32_t FDCAN_TxHeaderTypeDef::DataLength`**
Specifies the length of the frame that will be transmitted. This parameter can be a value of **FDCAN Data Length Code**
- **`uint32_t FDCAN_TxHeaderTypeDef::ErrorStateIndicator`**
Specifies the error state indicator. This parameter can be a value of **FDCAN Error State Indicator**
- **`uint32_t FDCAN_TxHeaderTypeDef::BitRateSwitch`**
Specifies whether the Tx frame will be transmitted with or without bit rate switching. This parameter can be a value of **FDCAN Bit Rate Switching**
- **`uint32_t FDCAN_TxHeaderTypeDef::FDFormat`**
Specifies whether the Tx frame will be transmitted in classic or FD format. This parameter can be a value of **FDCAN format**
- **`uint32_t FDCAN_TxHeaderTypeDef::TxEventFifoControl`**
Specifies the event FIFO control. This parameter can be a value of **Section 23.3.1 FDCAN**
- **`uint32_t FDCAN_TxHeaderTypeDef::MessageMarker`**
Specifies the message marker to be copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF

23.1.5

FDCAN_RxHeaderTypeDef

Data Fields

- **`uint32_t Identifier`**
- **`uint32_t IdType`**
- **`uint32_t RxFrameType`**
- **`uint32_t DataLength`**
- **`uint32_t ErrorStateIndicator`**
- **`uint32_t BitRateSwitch`**
- **`uint32_t FDFormat`**
- **`uint32_t RxTimestamp`**
- **`uint32_t FilterIndex`**
- **`uint32_t IsFilterMatchingFrame`**

Field Documentation

- **`uint32_t FDCAN_RxHeaderTypeDef::Identifier`**
Specifies the identifier. This parameter must be a number between:
 - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
 - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID

- **`uint32_t FDCAN_RxHeaderTypeDef::IdType`**
Specifies the identifier type of the received message. This parameter can be a value of **FDCAN ID Type**
- **`uint32_t FDCAN_RxHeaderTypeDef::RxFrameType`**
Specifies the the received message frame type. This parameter can be a value of **FDCAN Frame Type**
- **`uint32_t FDCAN_RxHeaderTypeDef::DataLength`**
Specifies the received frame length. This parameter can be a value of **FDCAN Data Length Code**
- **`uint32_t FDCAN_RxHeaderTypeDef::ErrorStateIndicator`**
Specifies the error state indicator. This parameter can be a value of **FDCAN Error State Indicator**
- **`uint32_t FDCAN_RxHeaderTypeDef::BitRateSwitch`**
Specifies whether the Rx frame is received with or without bit rate switching. This parameter can be a value of **FDCAN Bit Rate Switching**
- **`uint32_t FDCAN_RxHeaderTypeDef::FDFormat`**
Specifies whether the Rx frame is received in classic or FD format. This parameter can be a value of **FDCAN format**
- **`uint32_t FDCAN_RxHeaderTypeDef::RxTimestamp`**
Specifies the timestamp counter value captured on start of frame reception. This parameter must be a number between 0 and 0xFFFF
- **`uint32_t FDCAN_RxHeaderTypeDef::FilterIndex`**
Specifies the index of matching Rx acceptance filter element. This parameter must be a number between:
 - 0 and 127, if IdType is FDCAN_STANDARD_ID
 - 0 and 63, if IdType is FDCAN_EXTENDED_ID
- **`uint32_t FDCAN_RxHeaderTypeDef::IsFilterMatchingFrame`**
Specifies whether the accepted frame did not match any Rx filter. Acceptance of non-matching frames may be enabled via **HAL_FDCAN_ConfigGlobalFilter()**. This parameter can be 0 or 1

23.1.6 FDCAN_TxEventFifoTypeDef

Data Fields

- **`uint32_t Identifier`**
- **`uint32_t IdType`**
- **`uint32_t TxFrameType`**
- **`uint32_t DataLength`**
- **`uint32_t ErrorStateIndicator`**
- **`uint32_t BitRateSwitch`**
- **`uint32_t FDFormat`**
- **`uint32_t TxTimestamp`**
- **`uint32_t MessageMarker`**
- **`uint32_t EventType`**

Field Documentation

- **`uint32_t FDCAN_TxEventFifoTypeDef::Identifier`**
Specifies the identifier. This parameter must be a number between:
 - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
 - 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- **`uint32_t FDCAN_TxEventFifoTypeDef::IdType`**
Specifies the identifier type for the transmitted message. This parameter can be a value of **FDCAN ID Type**

- **`uint32_t FDCAN_TxEventFifoTypeDef::TxFrameType`**
Specifies the frame type of the transmitted message. This parameter can be a value of **FDCAN Frame Type**
- **`uint32_t FDCAN_TxEventFifoTypeDef::DataLength`**
Specifies the length of the transmitted frame. This parameter can be a value of **FDCAN Data Length Code**
- **`uint32_t FDCAN_TxEventFifoTypeDef::ErrorStateIndicator`**
Specifies the error state indicator. This parameter can be a value of **FDCAN Error State Indicator**
- **`uint32_t FDCAN_TxEventFifoTypeDef::BitRateSwitch`**
Specifies whether the Tx frame is transmitted with or without bit rate switching. This parameter can be a value of **FDCAN Bit Rate Switching**
- **`uint32_t FDCAN_TxEventFifoTypeDef::FDFormat`**
Specifies whether the Tx frame is transmitted in classic or FD format. This parameter can be a value of **FDCAN format**
- **`uint32_t FDCAN_TxEventFifoTypeDef::TxTimestamp`**
Specifies the timestamp counter value captured on start of frame transmission. This parameter must be a number between 0 and 0xFFFF
- **`uint32_t FDCAN_TxEventFifoTypeDef::MessageMarker`**
Specifies the message marker copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF
- **`uint32_t FDCAN_TxEventFifoTypeDef::EventType`**
Specifies the event type. This parameter can be a value of **FDCAN Event Type**

23.1.7 FDCAN_HpMsgStatusTypeDef

Data Fields

- **`uint32_t FilterList`**
- **`uint32_t FilterIndex`**
- **`uint32_t MessageStorage`**
- **`uint32_t MessageIndex`**

Field Documentation

- **`uint32_t FDCAN_HpMsgStatusTypeDef::FilterList`**
Specifies the filter list of the matching filter element. This parameter can be:
 - 0 : Standard Filter List
 - 1 : Extended Filter List
- **`uint32_t FDCAN_HpMsgStatusTypeDef::FilterIndex`**
Specifies the index of matching filter element. This parameter can be a number between:
 - 0 and 127, if FilterList is 0 (Standard)
 - 0 and 63, if FilterList is 1 (Extended)
- **`uint32_t FDCAN_HpMsgStatusTypeDef::MessageStorage`**
Specifies the HP Message Storage. This parameter can be a value of **FDCAN High Priority Message Storage**
- **`uint32_t FDCAN_HpMsgStatusTypeDef::MessageIndex`**
Specifies the Index of Rx FIFO element to which the message was stored. This parameter is valid only when MessageStorage is: FDCAN_HP_STORAGE_RXFIFO0 or FDCAN_HP_STORAGE_RXFIFO1

23.1.8 FDCAN_ProtocolStatusTypeDef

Data Fields

- `uint32_t LastErrorCode`
- `uint32_t DataLastErrorCode`
- `uint32_t Activity`
- `uint32_t ErrorPassive`
- `uint32_t Warning`
- `uint32_t BusOff`
- `uint32_t RxESIflag`
- `uint32_t RxBRSflag`
- `uint32_t RxFDFflag`
- `uint32_t ProtocolException`
- `uint32_t TDCvalue`

Field Documentation

- `uint32_t FDCAN_ProtocolStatusTypeDef::LastErrorCode`

Specifies the type of the last error that occurred on the FDCAN bus. This parameter can be a value of **FDCAN protocol error code**

- `uint32_t FDCAN_ProtocolStatusTypeDef::DataLastErrorCode`

Specifies the type of the last error that occurred in the data phase of a CAN FD format frame with its BRS flag set. This parameter can be a value of **FDCAN protocol error code**

- `uint32_t FDCAN_ProtocolStatusTypeDef::Activity`

Specifies the FDCAN module communication state. This parameter can be a value of **FDCAN communication state**

- `uint32_t FDCAN_ProtocolStatusTypeDef::ErrorPassive`

Specifies the FDCAN module error status. This parameter can be:

- 0 : The FDCAN is in Error_Active state
- 1 : The FDCAN is in Error_Passive state

- `uint32_t FDCAN_ProtocolStatusTypeDef::Warning`

Specifies the FDCAN module warning status. This parameter can be:

- 0 : error counters (RxErrorCnt and TxErrorCnt) are below the Error_Warning limit of 96
- 1 : at least one of error counters has reached the Error_Warning limit of 96

- `uint32_t FDCAN_ProtocolStatusTypeDef::BusOff`

Specifies the FDCAN module Bus_Off status. This parameter can be:

- 0 : The FDCAN is not in Bus_Off state
- 1 : The FDCAN is in Bus_Off state

- `uint32_t FDCAN_ProtocolStatusTypeDef::RxESIflag`

Specifies ESI flag of last received CAN FD message. This parameter can be:

- 0 : Last received CAN FD message did not have its ESI flag set
- 1 : Last received CAN FD message had its ESI flag set

- `uint32_t FDCAN_ProtocolStatusTypeDef::RxBRSflag`

Specifies BRS flag of last received CAN FD message. This parameter can be:

- 0 : Last received CAN FD message did not have its BRS flag set
- 1 : Last received CAN FD message had its BRS flag set

- `uint32_t FDCAN_ProtocolStatusTypeDef::RxFDFflag`

Specifies FDF flag of last received CAN FD message. This parameter can be:

- 0 : Last received CAN FD message did not have its FDF flag set
- 1 : Last received CAN FD message had its FDF flag set

- ***uint32_t FDCAN_ProtocolStatusTypeDef::ProtocolException***

Specifies the FDCAN module Protocol Exception status. This parameter can be:

- 0 : No protocol exception event occurred since last read access
- 1 : Protocol exception event occurred

- ***uint32_t FDCAN_ProtocolStatusTypeDef::TDCvalue***

Specifies the Transmitter Delay Compensation Value. This parameter can be a number between 0 and 127

23.1.9 FDCAN_ErrorCountersTypeDef

Data Fields

- ***uint32_t TxErrorCnt***
- ***uint32_t RxErrorCnt***
- ***uint32_t RxErrorPassive***
- ***uint32_t ErrorLogging***

Field Documentation

- ***uint32_t FDCAN_ErrorCountersTypeDef::TxErrorCnt***

Specifies the Transmit Error Counter Value. This parameter can be a number between 0 and 255

- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorCnt***

Specifies the Receive Error Counter Value. This parameter can be a number between 0 and 127

- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorPassive***

Specifies the Receive Error Passive status. This parameter can be:

- 0 : The Receive Error Counter (RxErrorCnt) is below the error passive level of 128
- 1 : The Receive Error Counter (RxErrorCnt) has reached the error passive level of 128

- ***uint32_t FDCAN_ErrorCountersTypeDef::ErrorLogging***

Specifies the Transmit/Receive error logging counter value. This parameter can be a number between 0 and 127. This counter is incremented each time when a FDCAN protocol error causes the TxErrorCnt or the RxErrorCnt to be incremented. The counter stops at 127; the next increment of TxErrorCnt or RxErrorCnt sets interrupt flag FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

23.1.10 FDCAN_TT_ConfigTypeDef

Data Fields

- ***uint32_t OperationMode***
- ***uint32_t GapEnable***
- ***uint32_t TimeMaster***
- ***uint32_t SyncDevLimit***
- ***uint32_t InitRefTrigOffset***
- ***uint32_t ExternalClkSync***
- ***uint32_t AppWdgLimit***
- ***uint32_t GlobalTimeFilter***
- ***uint32_t ClockCalibration***
- ***uint32_t EvtTrigPolarity***
- ***uint32_t BasicCyclesNbr***
- ***uint32_t CycleStartSync***
- ***uint32_t TxEnableWindow***

- `uint32_t ExpTxTrigNbr`
- `uint32_t TURNumerator`
- `uint32_t TURDenominator`
- `uint32_t TriggerMemoryNbr`
- `uint32_t StopWatchTrigSel`
- `uint32_t EventTrigSel`

Field Documentation

- `uint32_t FDCAN_TT_ConfigTypeDef::OperationMode`

Specifies the FDCAN Operation Mode. This parameter can be a value of **FDCAN Operation Mode**

- `uint32_t FDCAN_TT_ConfigTypeDef::GapEnable`

Specifies the FDCAN TT Operation. This parameter can be a value of **FDCAN TT Operation**. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0

- `uint32_t FDCAN_TT_ConfigTypeDef::TimeMaster`

Specifies whether the instance is a slave or a potential master. This parameter can be a value of **FDCAN TT Time Master**

- `uint32_t FDCAN_TT_ConfigTypeDef::SyncDevLimit`

Specifies the Synchronization Deviation Limit SDL of the TUR numerator : TUR = (Numerator ± SDL) / Denominator. With : SDL = 2^(SyncDevLimit+5). This parameter must be a number between 0 and 7

- `uint32_t FDCAN_TT_ConfigTypeDef::InitRefTrigOffset`

Specifies the Initial Reference Trigger Offset. This parameter must be a number between 0 and 127

- `uint32_t FDCAN_TT_ConfigTypeDef::ExternalClkSync`

Enable or disable External Clock Synchronization. This parameter can be a value of **FDCAN TT External Clock Synchronization**. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1

- `uint32_t FDCAN_TT_ConfigTypeDef::AppWdgLimit`

Specifies the Application Watchdog Limit : maximum time after which the application has to serve the application watchdog. The application watchdog is incremented once each 256 NTUs. The application watchdog can be disabled by setting AppWdgLimit to 0. This parameter must be a number between 0 and 255. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0

- `uint32_t FDCAN_TT_ConfigTypeDef::GlobalTimeFilter`

Enable or disable Global Time Filtering. This parameter can be a value of **FDCAN TT Global Time Filtering**. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1

- `uint32_t FDCAN_TT_ConfigTypeDef::ClockCalibration`

Enable or disable Automatic Clock Calibration. This parameter can be a value of **FDCAN TT Automatic Clock Calibration**. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL1

- `uint32_t FDCAN_TT_ConfigTypeDef::EvtTrigPolarity`

Specifies the Event Trigger Polarity. This parameter can be a value of **FDCAN TT Event Trigger Polarity**. This parameter is ignored if OperationMode is set to FDCAN_TT_COMMUNICATION_LEVEL0

- `uint32_t FDCAN_TT_ConfigTypeDef::BasicCyclesNbr`

Specifies the number of basic cycles in the system matrix. This parameter can be a value of **FDCAN TT Basic Cycle Number**

- `uint32_t FDCAN_TT_ConfigTypeDef::CycleStartSync`

Enable or disable synchronization pulse output at pin fdcan1_soc. This parameter can be a value of **FDCAN TT Cycle Start Sync**

- `uint32_t FDCAN_TT_ConfigTypeDef::TxEnableWindow`

- Specifies the length of Tx enable window in NTUs. This parameter must be a number between 1 and 16
- *uint32_t FDCAN_TT_ConfigTypeDef::ExpTxTrigNbr*
 - Specifies the number of expected Tx_Triggers in the system matrix. This is the sum of Tx_Triggers for exclusive, single arbitrating and merged arbitrating windows. This parameter must be a number between 0 and 4095
 - *uint32_t FDCAN_TT_ConfigTypeDef::TURNumerator*
 - Specifies the TUR (Time Unit Ratio) numerator. It is advised to set this parameter to the largest applicable value. This parameter must be a number between 0x10000 and 0x1FFF
 - *uint32_t FDCAN_TT_ConfigTypeDef::TURDenominator*
 - Specifies the TUR (Time Unit Ratio) denominator. This parameter must be a number between 0x0001 and 0x3FFF
 - *uint32_t FDCAN_TT_ConfigTypeDef::TriggerMemoryNbr*
 - Specifies the number of trigger memory elements. This parameter must be a number between 0 and 64
 - *uint32_t FDCAN_TT_ConfigTypeDef::StopWatchTrigSel*
 - Specifies the input to be used as stop watch trigger. This parameter can be a value of **FDCAN TT Stop Watch Trigger Selection**
 - *uint32_t FDCAN_TT_ConfigTypeDef::EventTrigSel*
 - Specifies the input to be used as event trigger. This parameter can be a value of **FDCAN TT Event Trigger Selection**

23.1.11 FDCAN_TriggerTypeDef

Data Fields

- *uint32_t TriggerIndex*
- *uint32_t TimeMark*
- *uint32_t RepeatFactor*
- *uint32_t StartCycle*
- *uint32_t TmEventInt*
- *uint32_t TmEventExt*
- *uint32_t TriggerType*
- *uint32_t FilterType*
- *uint32_t TxBufferIndex*
- *uint32_t FilterIndex*

Field Documentation

- *uint32_t FDCAN_TriggerTypeDef::TriggerIndex*

Specifies the trigger which will be configured. This parameter must be a number between 0 and 63
- *uint32_t FDCAN_TriggerTypeDef::TimeMark*

Specifies the cycle time for which the trigger becomes active. This parameter must be a number between 0 and 0xFFFF
- *uint32_t FDCAN_TriggerTypeDef::RepeatFactor*

Specifies the trigger repeat factor. This parameter can be a value of **FDCAN TT repeat factor**
- *uint32_t FDCAN_TriggerTypeDef::StartCycle*

Specifies the index of the first cycle in which the trigger becomes active. This parameter is ignored if RepeatFactor is set to FDCAN_TT_REPEAT_EVERY_CYCLE. This parameter must be a number between 0 and RepeatFactor
- *uint32_t FDCAN_TriggerTypeDef::TmEventInt*

Enable or disable the internal time mark event. If enabled, FDCAN_TT_FLAG_TRIG_TIME_MARK flag is set when trigger memory element becomes active. This parameter can be a value of **FDCAN TT time mark event internal**

- `uint32_t FDCAN_TriggerTypeDef::TmEventExt`

Enable or disable the external time mark event. If enabled, and if TTOCN.TTIE is set, a pulse is generated at fdcan1_tmp when trigger memory element becomes active. This parameter can be a value of **FDCAN TT time mark event external**

- `uint32_t FDCAN_TriggerTypeDef::TriggerType`

Specifies the trigger type. This parameter can be a value of **FDCAN TT trigger type**

- `uint32_t FDCAN_TriggerTypeDef::FilterType`

Specifies the filter identifier type. This parameter can be a value of **FDCAN ID Type**

- `uint32_t FDCAN_TriggerTypeDef::TxBufferIndex`

Specifies the index of the Tx buffer for which the trigger is valid. This parameter can be a value of **FDCAN Tx Location**. This parameter is taken in consideration only if the trigger is configured for transmission.

- `uint32_t FDCAN_TriggerTypeDef::FilterIndex`

Specifies the filter for which the trigger is valid. This parameter is taken in consideration only if the trigger is configured for reception. This parameter must be a number between:

- 0 and 127, if FilterType is FDCAN_STANDARD_ID
- 0 and 63, if FilterType is FDCAN_EXTENDED_ID

23.1.12 FDCAN_TTOperationStatusTypeDef

Data Fields

- `uint32_t ErrorLevel`
- `uint32_t MasterState`
- `uint32_t SyncState`
- `uint32_t GTimeQuality`
- `uint32_t ClockQuality`
- `uint32_t RefTrigOffset`
- `uint32_t GTimeDiscPending`
- `uint32_t GapFinished`
- `uint32_t MasterPriority`
- `uint32_t GapStarted`
- `uint32_t WaitForEvt`
- `uint32_t AppWdgEvt`
- `uint32_t ECSPending`
- `uint32_t PhaseLock`

Field Documentation

- `uint32_t FDCAN_TTOperationStatusTypeDef::ErrorLevel`

Specifies the type of the TT operation error level. This parameter can be a value of **FDCAN TT Error Level**

- `uint32_t FDCAN_TTOperationStatusTypeDef::MasterState`

Specifies the type of the TT master state. This parameter can be a value of **FDCAN TT Master State**

- `uint32_t FDCAN_TTOperationStatusTypeDef::SyncState`

Specifies the type of the TT synchronization state. This parameter can be a value of **FDCAN TT Synchronization State**

- `uint32_t FDCAN_TTOperationStatusTypeDef::GTimeQuality`

Specifies the Quality of Global Time Phase. This parameter is only relevant in Level 0 and Level 2, otherwise fixed to 0. This parameter can be:

- 0 : Global time not valid
- 1 : Global time in phase with Time Master

- ***uint32_t FDCAN_TTOperationStatusTypeDef::ClockQuality***

Specifies the Quality of Clock Speed. This parameter is only relevant in Level 0 and Level 2, otherwise fixed to 1. This parameter can be:

- 0 : Local clock speed not synchronized to Time Master clock speed
- 1 : Synchronization Deviation ≤ SDL

- ***uint32_t FDCAN_TTOperationStatusTypeDef::RefTrigOffset***

Specifies the Actual Reference Trigger Offset Value. This parameter can be a number between 0 and 0xFF

- ***uint32_t FDCAN_TTOperationStatusTypeDef::GTimeDiscPending***

Specifies the Global Time Discontinuity State. This parameter can be:

- 0 : No global time preset pending
- 1 : Node waits for the global time preset to take effect

- ***uint32_t FDCAN_TTOperationStatusTypeDef::GapFinished***

Specifies whether a Gap is finished. This parameter can be:

- 0 : Reset at the end of each reference message
- 1 : Gap finished

- ***uint32_t FDCAN_TTOperationStatusTypeDef::MasterPriority***

Specifies the Priority of actual Time Master. This parameter can be a number between 0 and 0x7

- ***uint32_t FDCAN_TTOperationStatusTypeDef::GapStarted***

Specifies whether a Gap is started. This parameter can be:

- 0 : No Gap in schedule
- 1 : Gap time after Basic Cycle has started

- ***uint32_t FDCAN_TTOperationStatusTypeDef::WaitForEvt***

Specifies whether a Gap is announced. This parameter can be:

- 0 : No Gap announced, reset by a reference message with Next_is_Gap = 0
- 1 : Reference message with Next_is_Gap = 1 received

- ***uint32_t FDCAN_TTOperationStatusTypeDef::AppWdgEvt***

Specifies the Application Watchdog State. This parameter can be:

- 0 : Application Watchdog served in time
- 1 : Failed to serve Application Watchdog in time

- ***uint32_t FDCAN_TTOperationStatusTypeDef::ECSPending***

Specifies the External Clock Synchronization State. This parameter can be:

- 0 : No external clock synchronization pending
- 1 : Node waits for external clock synchronization to take effect

- ***uint32_t FDCAN_TTOperationStatusTypeDef::PhaseLock***

Specifies the Phase Lock State. This parameter can be:

- 0 : Phase outside range
- 1 : Phase inside range

23.1.13 FDCAN_MsgRamAddressTypeDef

Data Fields

- ***uint32_t StandardFilterSA***

- `uint32_t ExtendedFilterSA`
- `uint32_t RxFIFO0SA`
- `uint32_t RxFIFO1SA`
- `uint32_t RxBufferSA`
- `uint32_t TxEventFIFOSA`
- `uint32_t TxBUFFERSA`
- `uint32_t TxFIFOQSA`
- `uint32_t TTMemorySA`
- `uint32_t EndAddress`

Field Documentation

- `uint32_t FDCAN_MsgRamAddressTypeDef::StandardFilterSA`

Specifies the Standard Filter List Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::ExtendedFilterSA`

Specifies the Extended Filter List Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO0SA`

Specifies the Rx FIFO 0 Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO1SA`

Specifies the Rx FIFO 1 Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::RxBufferSA`

Specifies the Rx Buffer Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::TxEventFIFOSA`

Specifies the Tx Event FIFO Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::TxBUFFERSA`

Specifies the Tx Buffers Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::TxFIFOQSA`

Specifies the Tx FIFO/Queue Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::TTMemorySA`

Specifies the Trigger Memory Start Address. This parameter must be a 32-bit word address

- `uint32_t FDCAN_MsgRamAddressTypeDef::EndAddress`

Specifies the End Address of the allocated RAM. This parameter must be a 32-bit word address

23.1.14 FDCAN_HandleTypeDef

Data Fields

- `FDCAN_GlobalTypeDef * Instance`
- `TTCAN_TypeDef * ttcan`
- `FDCAN_InitTypeDef Init`
- `FDCAN_MsgRamAddressTypeDef msgRam`
- `_IO HAL_FDCAN_StateTypeDef State`
- `HAL_LockTypeDef Lock`
- `_IO uint32_t ErrorCode`

Field Documentation

- `FDCAN_GlobalTypeDef* FDCAN_HandleTypeDef::Instance`

Register base address

- ***TTCAN_TypeDef* FDCAN_HandleTypeDefTypeDef::ttcan***
TT register base address
- ***FDCAN_InitTypeDef FDCAN_HandleTypeDefTypeDef::Init***
FDCAN required parameters
- ***FDCAN_MsgRamAddressTypeDef FDCAN_HandleTypeDefTypeDef::msgRam***
FDCAN Message RAM blocks
- ***_IO HAL_FDCAN_StateTypeDef FDCAN_HandleTypeDefTypeDef::State***
FDCAN communication state
- ***HAL_LockTypeDef FDCAN_HandleTypeDefTypeDef::Lock***
FDCAN locking object
- ***_IO uint32_t FDCAN_HandleTypeDefTypeDef::ErrorCode***
FDCAN Error code

23.2 FDCAN Firmware driver API description

23.2.1 How to use this driver

1. Initialize the FDCAN peripheral using `HAL_FDCAN_Init` function.
2. If needed , configure the reception filters and optional features using the following configuration functions:
 - `HAL_FDCAN_ConfigClockCalibration`
 - `HAL_FDCAN_ConfigFilter`
 - `HAL_FDCAN_ConfigGlobalFilter`
 - `HAL_FDCAN_ConfigExtendedIdMask`
 - `HAL_FDCAN_ConfigRx_fifoOverwrite`
 - `HAL_FDCAN_ConfigFifoWatermark`
 - `HAL_FDCAN_ConfigRamWatchdog`
 - `HAL_FDCAN_ConfigTimestampCounter`
 - `HAL_FDCAN_EnableTimestampCounter`
 - `HAL_FDCAN_DisableTimestampCounter`
 - `HAL_FDCAN_ConfigTimeoutCounter`
 - `HAL_FDCAN_EnableTimeoutCounter`
 - `HAL_FDCAN_DisableTimeoutCounter`
 - `HAL_FDCAN_ConfigTxDelayCompensation`
 - `HAL_FDCAN_EnableTxDelayCompensation`
 - `HAL_FDCAN_DisableTxDelayCompensation`
 - `HAL_FDCAN_TT_ConfigOperation`
 - `HAL_FDCAN_TT_ConfigReferenceMessage`
 - `HAL_FDCAN_TT_ConfigTrigger`
3. Start the FDCAN module using `HAL_FDCAN_Start` function. At this level the node is active on the bus: it can send and receive messages.
4. The following Tx control functions can only be called when the FDCAN module is started:
 - `HAL_FDCAN_AddMessageToTxFifoQ`
 - `HAL_FDCAN_EnableTxBufferRequest`
 - `HAL_FDCAN_AbortTxRequest`
5. When a message is received into the FDCAN message RAM, it can be retrieved using the `HAL_FDCAN_GetRxMessage` function.

6. Calling the HAL_FDCAN_Stop function stops the FDCAN module by entering it to initialization mode and re-enabling access to configuration registers through the configuration functions listed here above.
7. All other control functions can be called any time after initialization phase, no matter if the FDCAN module is started or stopped.

Polling mode operation

1. Reception and transmission states can be monitored via the following functions:
 - HAL_FDCAN_IsRxBufferMessageAvailable
 - HAL_FDCAN_IsTxBufferMessagePending
 - HAL_FDCAN_GetRxFifoFillLevel
 - HAL_FDCAN_GetTxFifoFreeLevel

Interrupt mode operation

1. There are two interrupt lines: line 0 and 1. By default, all interrupts are assigned to line 0. Interrupt lines can be configured using HAL_FDCAN_ConfigInterruptLines function.
2. Notifications are activated using HAL_FDCAN_ActivateNotification function. Then, the process can be controlled through one of the available user callbacks: HAL_FDCAN_xxxCallback.

23.2.2

Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the FDCAN.
- De-initialize the FDCAN.
- Enter FDCAN peripheral in power down mode.
- Exit power down mode.

This section contains the following APIs:

- [**HAL_FDCAN_Init**](#)
- [**HAL_FDCAN_DelInit**](#)
- [**HAL_FDCAN_MspInit**](#)
- [**HAL_FDCAN_MspDelInit**](#)
- [**HAL_FDCAN_EnterPowerDownMode**](#)
- [**HAL_FDCAN_ExitPowerDownMode**](#)

23.2.3

Configuration functions

This section provides functions allowing to:

- HAL_FDCAN_ConfigClockCalibration : Configure the FDCAN clock calibration unit
- HAL_FDCAN_GetClockCalibrationState : Get the clock calibration state
- HAL_FDCAN_ResetClockCalibrationState : Reset the clock calibration state
- HAL_FDCAN_GetClockCalibrationCounter : Get the clock calibration counters values
- HAL_FDCAN_ConfigFilter : Configure the FDCAN reception filters
- HAL_FDCAN_ConfigGlobalFilter : Configure the FDCAN global filter
- HAL_FDCAN_ConfigExtendedIdMask : Configure the extended ID mask
- HAL_FDCAN_ConfigRx_fifoOverwrite : Configure the Rx FIFO operation mode
- HAL_FDCAN_ConfigFifoWatermark : Configure the FIFO watermark
- HAL_FDCAN_ConfigRamWatchdog : Configure the RAM watchdog
- HAL_FDCAN_ConfigTimestampCounter : Configure the timestamp counter
- HAL_FDCAN_EnableTimestampCounter : Enable the timestamp counter
- HAL_FDCAN_DisableTimestampCounter : Disable the timestamp counter
- HAL_FDCAN_GetTimestampCounter : Get the timestamp counter value

- HAL_FDCAN_ResetTimestampCounter : Reset the timestamp counter to zero
- HAL_FDCAN_ConfigTimeoutCounter : Configure the timeout counter
- HAL_FDCAN_EnableTimeoutCounter : Enable the timeout counter
- HAL_FDCAN_DisableTimeoutCounter : Disable the timeout counter
- HAL_FDCAN_GetTimeoutCounter : Get the timeout counter value
- HAL_FDCAN_ResetTimeoutCounter : Reset the timeout counter to its start value
- HAL_FDCAN_ConfigTxDelayCompensation : Configure the transmitter delay compensation
- HAL_FDCAN_EnableTxDelayCompensation : Enable the transmitter delay compensation
- HAL_FDCAN_DisableTxDelayCompensation : Disable the transmitter delay compensation

This section contains the following APIs:

- [HAL_FDCAN_ConfigClockCalibration](#)
- [HAL_FDCAN_GetClockCalibrationState](#)
- [HAL_FDCAN_ResetClockCalibrationState](#)
- [HAL_FDCAN_GetClockCalibrationCounter](#)
- [HAL_FDCAN_ConfigFilter](#)
- [HAL_FDCAN_ConfigGlobalFilter](#)
- [HAL_FDCAN_ConfigExtendedIdMask](#)
- [HAL_FDCAN_ConfigRx_fifoOverwrite](#)
- [HAL_FDCAN_ConfigFifoWatermark](#)
- [HAL_FDCAN_ConfigRamWatchdog](#)
- [HAL_FDCAN_ConfigTimestampCounter](#)
- [HAL_FDCAN_EnableTimestampCounter](#)
- [HAL_FDCAN_DisableTimestampCounter](#)
- [HAL_FDCAN_GetTimestampCounter](#)
- [HAL_FDCAN_ResetTimestampCounter](#)
- [HAL_FDCAN_ConfigTimeoutCounter](#)
- [HAL_FDCAN_EnableTimeoutCounter](#)
- [HAL_FDCAN_DisableTimeoutCounter](#)
- [HAL_FDCAN_GetTimeoutCounter](#)
- [HAL_FDCAN_ResetTimeoutCounter](#)
- [HAL_FDCAN_ConfigTxDelayCompensation](#)
- [HAL_FDCAN_EnableTxDelayCompensation](#)
- [HAL_FDCAN_DisableTxDelayCompensation](#)

23.2.4

Control functions

This section provides functions allowing to:

- HAL_FDCAN_Start : Start the FDCAN module
- HAL_FDCAN_Stop : Stop the FDCAN module and enable access to configuration registers
- HAL_FDCAN_AddMessageToTxFifoQ : Add a message to the Tx FIFO/Queue and activate the corresponding transmission request
- HAL_FDCAN_AddMessageToTxBuffer : Add a message to a dedicated Tx buffer
- HAL_FDCAN_EnableTxBufferRequest : Enable transmission request
- HAL_FDCAN_AbortTxRequest : Abort transmission request
- HAL_FDCAN_GetRxMessage : Get an FDCAN frame from the Rx Buffer/FIFO zone into the message RAM
- HAL_FDCAN_GetTxEvent : Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM
- HAL_FDCAN_GetHighPriorityMessageStatus : Get high priority message status
- HAL_FDCAN_GetProtocolStatus : Get protocol status
- HAL_FDCAN_GetErrorCounters : Get error counter values

- `HAL_FDCAN_IsRxBufferMessageAvailable` : Check if a new message is received in the selected Rx buffer
- `HAL_FDCAN_IsTxBufferMessagePending` : Check if a transmission request is pending on the selected Tx buffer
- `HAL_FDCAN_GetRx_fifoFillLevel` : Return Rx FIFO fill level
- `HAL_FDCAN_GetTx_fifoFreeLevel` : Return Tx FIFO free level
- `HAL_FDCAN_IsRestrictedOperationMode` : Check if the FDCAN peripheral entered Restricted Operation Mode
- `HAL_FDCAN_ExitRestrictedOperationMode` : Exit Restricted Operation Mode

This section contains the following APIs:

- [`HAL_FDCAN_Start`](#)
- [`HAL_FDCAN_Stop`](#)
- [`HAL_FDCAN_AddMessageToTx_fifoQ`](#)
- [`HAL_FDCAN_AddMessageToTxBuffer`](#)
- [`HAL_FDCAN_EnableTxBufferRequest`](#)
- [`HAL_FDCAN_AbortTxRequest`](#)
- [`HAL_FDCAN_GetRxMessage`](#)
- [`HAL_FDCAN_GetTxEvent`](#)
- [`HAL_FDCAN_GetHighPriorityMessageStatus`](#)
- [`HAL_FDCAN_GetProtocolStatus`](#)
- [`HAL_FDCAN_GetErrorCounters`](#)
- [`HAL_FDCAN_IsRxBufferMessageAvailable`](#)
- [`HAL_FDCAN_IsTxBufferMessagePending`](#)
- [`HAL_FDCAN_GetRx_fifoFillLevel`](#)
- [`HAL_FDCAN_GetTx_fifoFreeLevel`](#)
- [`HAL_FDCAN_IsRestrictedOperationMode`](#)
- [`HAL_FDCAN_ExitRestrictedOperationMode`](#)

23.2.5 TT Configuration and control functions

This section provides functions allowing to:

- `HAL_FDCAN_TT_ConfigOperation` : Initialize TT operation parameters
- `HAL_FDCAN_TT_ConfigReferenceMessage` : Configure the reference message
- `HAL_FDCAN_TT_ConfigTrigger` : Configure the FDCAN trigger
- `HAL_FDCAN_TT_SetGlobalTime` : Schedule global time adjustment
- `HAL_FDCAN_TT_SetClockSynchronization` : Schedule TUR numerator update
- `HAL_FDCAN_TT_ConfigStopWatch` : Configure stop watch source and polarity
- `HAL_FDCAN_TT_ConfigRegisterTimeMark` : Configure register time mark pulse generation
- `HAL_FDCAN_TT_EnableRegisterTimeMarkPulse` : Enable register time mark pulse generation
- `HAL_FDCAN_TT_DisableRegisterTimeMarkPulse` : Disable register time mark pulse generation
- `HAL_FDCAN_TT_EnableTriggerTimeMarkPulse` : Enable trigger time mark pulse generation
- `HAL_FDCAN_TT_DisableTriggerTimeMarkPulse` : Disable trigger time mark pulse generation
- `HAL_FDCAN_TT_EnableHardwareGapControl` : Enable gap control by input pin fdcan1_evt
- `HAL_FDCAN_TT_DisableHardwareGapControl` : Disable gap control by input pin fdcan1_evt
- `HAL_FDCAN_TT_EnableTimeMarkGapControl` : Enable gap control (finish only) by register time mark interrupt
- `HAL_FDCAN_TT_DisableTimeMarkGapControl` : Disable gap control by register time mark interrupt
- `HAL_FDCAN_TT_SetNextIsGap` : Transmit next reference message with Next_is_Gap="1"
- `HAL_FDCAN_TT_SetEndOfGap` : Finish a Gap by requesting start of reference message
- `HAL_FDCAN_TT_ConfigExternalSyncPhase` : Configure target phase used for external synchronization

- HAL_FDCAN_TT_EnableExternalSynchronization : Synchronize the phase of the FDCAN schedule to an external schedule
- HAL_FDCAN_TT_DisableExternalSynchronization : Disable external schedule synchronization
- HAL_FDCAN_TT_GetOperationStatus : Get TT operation status

This section contains the following APIs:

- [HAL_FDCAN_TT_ConfigOperation](#)
- [HAL_FDCAN_TT_ConfigReferenceMessage](#)
- [HAL_FDCAN_TT_ConfigTrigger](#)
- [HAL_FDCAN_TT_SetGlobalTime](#)
- [HAL_FDCAN_TT_SetClockSynchronization](#)
- [HAL_FDCAN_TT_ConfigStopWatch](#)
- [HAL_FDCAN_TT_ConfigRegisterTimeMark](#)
- [HAL_FDCAN_TT_EnableRegisterTimeMarkPulse](#)
- [HAL_FDCAN_TT_DisableRegisterTimeMarkPulse](#)
- [HAL_FDCAN_TT_EnableTriggerTimeMarkPulse](#)
- [HAL_FDCAN_TT_DisableTriggerTimeMarkPulse](#)
- [HAL_FDCAN_TT_EnableHardwareGapControl](#)
- [HAL_FDCAN_TT_DisableHardwareGapControl](#)
- [HAL_FDCAN_TT_EnableTimeMarkGapControl](#)
- [HAL_FDCAN_TT_DisableTimeMarkGapControl](#)
- [HAL_FDCAN_TT_SetNextIsGap](#)
- [HAL_FDCAN_TT_SetEndOfGap](#)
- [HAL_FDCAN_TT_ConfigExternalSyncPhase](#)
- [HAL_FDCAN_TT_EnableExternalSynchronization](#)
- [HAL_FDCAN_TT_DisableExternalSynchronization](#)
- [HAL_FDCAN_TT_GetOperationStatus](#)

23.2.6

Interrupts management

This section provides functions allowing to:

- HAL_FDCAN_ConfigInterruptLines : Assign interrupts to either Interrupt line 0 or 1
- HAL_FDCAN_TT_ConfigInterruptLines : Assign TT interrupts to either Interrupt line 0 or 1
- HAL_FDCAN_ActivateNotification : Enable interrupts
- HAL_FDCAN_DeactivateNotification : Disable interrupts
- HAL_FDCAN_TT_ActivateNotification : Enable TT interrupts
- HAL_FDCAN_TT_DeactivateNotification : Disable TT interrupts
- HAL_FDCAN_IRQHandler : Handles FDCAN interrupt request

This section contains the following APIs:

- [HAL_FDCAN_ConfigInterruptLines](#)
- [HAL_FDCAN_TT_ConfigInterruptLines](#)
- [HAL_FDCAN_ActivateNotification](#)
- [HAL_FDCAN_DeactivateNotification](#)
- [HAL_FDCAN_TT_ActivateNotification](#)
- [HAL_FDCAN_TT_DeactivateNotification](#)
- [HAL_FDCAN_IRQHandler](#)

23.2.7

Callback functions

This subsection provides the following callback functions:

- HAL_FDCAN_ClockCalibrationCallback
- HAL_FDCAN_TxEventFifoCallback
- HAL_FDCAN_RxFifo0Callback
- HAL_FDCAN_RxFifo1Callback
- HAL_FDCAN_TxFifoEmptyCallback
- HAL_FDCAN_TxBufferCompleteCallback
- HAL_FDCAN_TxBufferAbortCallback
- HAL_FDCAN_RxBufferNewMessageCallback
- HAL_FDCAN_HighPriorityMessageCallback
- HAL_FDCAN_TimestampWraparoundCallback
- HAL_FDCAN_TimeoutOccurredCallback
- HAL_FDCAN_ErrorCallback
- HAL_FDCAN_TTSchedSyncCallback
- HAL_FDCAN_TTTimeMarkCallback
- HAL_FDCAN_TTStopWatchCallback
- HAL_FDCAN_TTGlobalTimeCallback

This section contains the following APIs:

- [HAL_FDCAN_ClockCalibrationCallback](#)
- [HAL_FDCAN_TxEventFifoCallback](#)
- [HAL_FDCAN_RxFifo0Callback](#)
- [HAL_FDCAN_RxFifo1Callback](#)
- [HAL_FDCAN_TxFifoEmptyCallback](#)
- [HAL_FDCAN_TxBufferCompleteCallback](#)
- [HAL_FDCAN_TxBufferAbortCallback](#)
- [HAL_FDCAN_RxBufferNewMessageCallback](#)
- [HAL_FDCAN_TimestampWraparoundCallback](#)
- [HAL_FDCAN_TimeoutOccurredCallback](#)
- [HAL_FDCAN_HighPriorityMessageCallback](#)
- [HAL_FDCAN_ErrorCallback](#)
- [HAL_FDCAN_TT_ScheduleSyncCallback](#)
- [HAL_FDCAN_TT_TimeMarkCallback](#)
- [HAL_FDCAN_TT_StopWatchCallback](#)
- [HAL_FDCAN_TT_GlobalTimeCallback](#)

23.2.8 Peripheral State functions

This subsection provides functions allowing to :

- HAL_FDCAN_GetState() : Return the FDCAN state.
- HAL_FDCAN_GetError() : Return the FDCAN error code if any.

This section contains the following APIs:

- [HAL_FDCAN_GetState](#)
- [HAL_FDCAN_GetError](#)

23.2.9 Detailed description of functions

HAL_FDCAN_Init

Function name

`HAL_StatusTypeDef HAL_FDCAN_Init (FDCAN_HandleTypeDef * hfdcan)`

Function description

Initializes the FDCAN peripheral according to the specified parameters in the FDCAN_InitTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DeInit

Function name

HAL_StatusTypeDef HAL_FDCAN_DeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Deinitializes the FDCAN peripheral registers to their default reset values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_MspInit

Function name

void HAL_FDCAN_MspInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_MspDeInit

Function name

void HAL_FDCAN_MspDeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Deinitializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_EnterPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnterPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enter FDCAN peripheral in sleep mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ExitPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit power down mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigClockCalibration

Function name

**HAL_StatusTypeDef HAL_FDCAN_ConfigClockCalibration (FDCAN_HandleTypeDef * hfdcan,
FDCAN_ClkCalUnitTypeDef * sCcuConfig)**

Function description

Configure the FDCAN clock calibration unit according to the specified parameters in the FDCAN_ClkCalUnitTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **sCcuConfig:** pointer to an FDCAN_ClkCalUnitTypeDef structure that contains the clock calibration information

Return values

- **HAL:** status

HAL_FDCAN_GetClockCalibrationState

Function name

uint32_t HAL_FDCAN_GetClockCalibrationState (FDCAN_HandleTypeDef * hfdcan)

Function description

Get the clock calibration state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **State:** clock calibration state (can be a value of
 - FDCAN_calibration_state)

HAL_FDCAN_ResetClockCalibrationState

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetClockCalibrationState (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Reset the clock calibration state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetClockCalibrationCounter

Function name

uint32_t HAL_FDCAN_GetClockCalibrationCounter (FDCAN_HandleTypeDefDef * hfdcan, uint32_t Counter)

Function description

Get the clock calibration counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.
- **Counter:** clock calibration counter. This parameter can be a value of
 - FDCAN_calibration_counter.

Return values

- **Value:** clock calibration counter value

HAL_FDCAN_ConfigFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigFilter (FDCAN_HandleTypeDefDef * hfdcan, FDCAN_FilterTypeDef * sFilterConfig)

Function description

Configure the FDCAN reception filter according to the specified parameters in the FDCAN_FilterTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.
- **sFilterConfig:** pointer to an FDCAN_FilterTypeDef structure that contains the filter configuration information

Return values

- **HAL:** status

HAL_FDCAN_ConfigGlobalFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigGlobalFilter (FDCAN_HandleTypeDef * hfdcan, uint32_t NonMatchingStd, uint32_t NonMatchingExt, uint32_t RejectRemoteStd, uint32_t RejectRemoteExt)

Function description

Configure the FDCAN global filter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **NonMatchingStd:** Defines how received messages with 11-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
 - FDCAN_Non_Matching_Frames.
- **NonMatchingExt:** Defines how received messages with 29-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
 - FDCAN_Non_Matching_Frames.
- **RejectRemoteStd:** Enable or disable the remote standard frames rejection. This parameter can be set to ENABLE or DISABLE.
- **RejectRemoteExt:** Enable or disable the remote extended frames rejection. This parameter can be set to ENABLE or DISABLE.

Return values

- **HAL:** status

HAL_FDCAN_ConfigExtendedIdMask

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigExtendedIdMask (FDCAN_HandleTypeDef * hfdcan, uint32_t Mask)

Function description

Configure the extended ID mask.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **Mask:** Extended ID Mask. This parameter must be a number between 0 and 0xFFFFFFFF

Return values

- **HAL:** status

HAL_FDCAN_ConfigRxFifoOverwrite

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRxFifoOverwrite (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo, uint32_t OperationMode)

Function description

Configure the Rx FIFO operation mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
 - FDCAN_RX_FIFO0: Rx FIFO 0
 - FDCAN_RX_FIFO1: Rx FIFO 1
- **OperationMode:** operation mode. This parameter can be a value of
 - FDCAN_Rx_FIFO_operation_mode.

Return values

- **HAL:** status

HAL_FDCAN_ConfigFifoWatermark

Function name

```
HAL_StatusTypeDef HAL_FDCAN_ConfigFifoWatermark (FDCAN_HandleTypeDef * hfdcan, uint32_t t  
FIFO, uint32_t Watermark)
```

Function description

Configure the FIFO watermark.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **FIFO:** select the FIFO to be configured. This parameter can be a value of
 - FDCAN_FIFO_watermark.
- **Watermark:** level for FIFO watermark interrupt. This parameter must be a number between:
 - 0 and 32, if FIFO is FDCAN_CFG_TX_EVENT_FIFO
 - 0 and 64, if FIFO is FDCAN_CFG_RX_FIFO0 or FDCAN_CFG_RX_FIFO1

Return values

- **HAL:** status

HAL_FDCAN_ConfigRamWatchdog

Function name

```
HAL_StatusTypeDef HAL_FDCAN_ConfigRamWatchdog (FDCAN_HandleTypeDef * hfdcan, uint32_t  
CounterStartValue)
```

Function description

Configure the RAM watchdog.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **CounterStartValue:** Start value of the Message RAM Watchdog Counter, This parameter must be a number between 0x00 and 0xFF, with the reset value of 0x00 the counter is disabled.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimestampCounter

Function name

```
HAL_StatusTypeDef HAL_FDCAN_ConfigTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t  
TimestampPrescaler)
```

Function description

Configure the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.
- **TimestampPrescaler:** Timestamp Counter Prescaler. This parameter can be a value of
 - FDCAN_Timestamp_Prescaler.

Return values

- **HAL:** status

HAL_FDCAN_EnableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimestampCounter (FDCAN_HandleTypeDefDef * hfdcan, uint32_t TimestampOperation)

Function description

Enable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.
- **TimestampOperation:** Timestamp counter operation. This parameter can be a value of
 - FDCAN_Timestamp.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimestampCounter (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Disable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimestampCounter

Function name

uint16_t HAL_FDCAN_GetTimestampCounter (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Get the timestamp counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Value:** Timestamp counter value

HAL_FDCAN_ResetTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timestamp counter to zero.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimeoutCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimeoutOperation, uint32_t TimeoutPeriod)

Function description

Configure the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimeoutOperation:** Timeout counter operation. This parameter can be a value of
 - FDCAN_Timeout_Operation.
- **TimeoutPeriod:** Start value of the timeout down-counter. This parameter must be a number between 0x0000 and 0xFFFF

Return values

- **HAL:** status

HAL_FDCAN_EnableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimeoutCounter

Function name

```
uint16_t HAL_FDCAN_GetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)
```

Function description

Get the timeout counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Value:** Timeout counter value

HAL_FDCAN_ResetTimeoutCounter

Function name

```
HAL_StatusTypeDef HAL_FDCAN_ResetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)
```

Function description

Reset the timeout counter to its start value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTxDelayCompensation

Function name

```
HAL_StatusTypeDef HAL_FDCAN_ConfigTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan,  
uint32_t TdcOffset, uint32_t TdcFilter)
```

Function description

Configure the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TdcOffset:** Transmitter Delay Compensation Offset. This parameter must be a number between 0x00 and 0xFF.
- **TdcFilter:** Transmitter Delay Compensation Filter Window Length. This parameter must be a number between 0x00 and 0xFF.

Return values

- **HAL:** status

HAL_FDCAN_EnableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Start

Function name

HAL_StatusTypeDef HAL_FDCAN_Start (FDCAN_HandleTypeDef * hfdcan)

Function description

Start the FDCAN module.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Stop

Function name

HAL_StatusTypeDef HAL_FDCAN_Stop (FDCAN_HandleTypeDef * hfdcan)

Function description

Stop the FDCAN module and enable access to configuration registers.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_AddMessageToTxFifoQ

Function name

**HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxFifoQ (FDCAN_HandleTypeDef * hfdcan,
FDCAN_TxHeaderTypeDef * pTxHeader, uint8_t * pTxData)**

Function description

Add a message to the Tx FIFO/Queue and activate the corresponding transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxHeader:** pointer to a FDCAN_TxHeaderTypeDef structure.
- **pTxData:** pointer to a buffer containing the payload of the Tx frame.

Return values

- **HAL:** status

HAL_FDCAN_AddMessageToTxBuffer

Function name

**HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxBuffer (FDCAN_HandleTypeDef * hfdcan,
FDCAN_TxHeaderTypeDef * pTxHeader, uint8_t * pTxData, uint32_t BufferIndex)**

Function description

Add a message to a dedicated Tx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxHeader:** pointer to a FDCAN_TxHeaderTypeDef structure.
- **pTxData:** pointer to a buffer containing the payload of the Tx frame.
- **BufferIndex:** index of the buffer to be configured. This parameter can be a value of
 - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_EnableTxBufferRequest

Function name

**HAL_StatusTypeDef HAL_FDCAN_EnableTxBufferRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t
BufferIndex)**

Function description

Enable transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndex:** buffer index. This parameter can be any combination of
 - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_AbortTxRequest

Function name

```
HAL_StatusTypeDef HAL_FDCAN_AbortTxRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t  
BufferIndex)
```

Function description

Abort transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndex:** buffer index. This parameter can be any combination of
 - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_GetRxMessage

Function name

```
HAL_StatusTypeDef HAL_FDCAN_GetRxMessage (FDCAN_HandleTypeDef * hfdcan, uint32_t  
RxLocation, FDCAN_RxHeaderTypeDef * pRxHeader, uint8_t * pRxData)
```

Function description

Get an FDCAN frame from the Rx Buffer/FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxLocation:** Location of the received message to be read. This parameter can be a value of
 - FDCAN_Rx_location.
- **pRxHeader:** pointer to a FDCAN_RxHeaderTypeDef structure.
- **pRxData:** pointer to a buffer where the payload of the Rx frame will be stored.

Return values

- **HAL:** status

HAL_FDCAN_GetTxEvent

Function name

```
HAL_StatusTypeDef HAL_FDCAN_GetTxEvent (FDCAN_HandleTypeDef * hfdcan,  
FDCAN_TxEventFifoTypeDef * pTxEvent)
```

Function description

Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxEvent:** pointer to a FDCAN_TxEventFifoTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetHighPriorityMessageStatus

Function name

```
HAL_StatusTypeDef HAL_FDCAN_GetHighPriorityMessageStatus (FDCAN_HandleTypeDef * hfdcan,  
FDCAN_HpMsgStatusTypeDef * HpMsgStatus)
```

Function description

Get high priority message status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **HpMsgStatus:** pointer to an FDCAN_HpMsgStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetProtocolStatus

Function name

```
HAL_StatusTypeDef HAL_FDCAN_GetProtocolStatus (FDCAN_HandleTypeDef * hfdcan,  
FDCAN_ProtocolStatusTypeDef * ProtocolStatus)
```

Function description

Get protocol status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ProtocolStatus:** pointer to an FDCAN_ProtocolStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetErrorCounters

Function name

```
HAL_StatusTypeDef HAL_FDCAN_GetErrorCounters (FDCAN_HandleTypeDef * hfdcan,  
FDCAN_ErrorCountersTypeDef * ErrorCounters)
```

Function description

Get error counter values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorCounters:** pointer to an FDCAN_ErrorCountersTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_IsRxBufferMessageAvailable

Function name

```
uint32_t HAL_FDCAN_IsRxBufferMessageAvailable (FDCAN_HandleTypeDef * hfdcan, uint32_t RxBufferIndex)
```

Function description

Check if a new message is received in the selected Rx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxBufferIndex:** Rx buffer index. This parameter must be a number between 0 and 63.

Return values

- **Status:**
 - 0 : No new message on RxBufferIndex.
 - 1 : New message received on RxBufferIndex.

HAL_FDCAN_IsTxBufferMessagePending

Function name

```
uint32_t HAL_FDCAN_IsTxBufferMessagePending (FDCAN_HandleTypeDef * hfdcan, uint32_t TxBufferIndex)
```

Function description

Check if a transmission request is pending on the selected Tx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxBufferIndex:** Tx buffer index. This parameter can be a value of
 - FDCAN_Tx_location.

Return values

- **Status:**
 - 0 : No pending transmission request on RxBufferIndex.
 - 1 : Pending transmission request on RxBufferIndex.

HAL_FDCAN_GetRx_fifoFillLevel

Function name

```
uint32_t HAL_FDCAN_GetRx_fifoFillLevel (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo)
```

Function description

Return Rx FIFO fill level.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
 - FDCAN_RX_FIFO0: Rx FIFO 0
 - FDCAN_RX_FIFO1: Rx FIFO 1

Return values

- **Level:** Rx FIFO fill level.

HAL_FDCAN_GetTxFifoFreeLevel

Function name

```
uint32_t HAL_FDCAN_GetTxFifoFreeLevel (FDCAN_HandleTypeDef * hfdcan)
```

Function description

Return Tx FIFO free level: number of consecutive free Tx FIFO elements starting from Tx FIFO GetIndex.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Level:** Tx FIFO free level.

HAL_FDCAN_IsRestrictedOperationMode

Function name

```
uint32_t HAL_FDCAN_IsRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)
```

Function description

Check if the FDCAN peripheral entered Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Status:**
 - 0 : Normal FDCAN operation.
 - 1 : Restricted Operation Mode active.

HAL_FDCAN_ExitRestrictedOperationMode

Function name

```
HAL_StatusTypeDef HAL_FDCAN_ExitRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)
```

Function description

Exit Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigOperation

Function name

```
HAL_StatusTypeDef HAL_FDCAN_TT_ConfigOperation (FDCAN_HandleTypeDef * hfdcan,  
FDCAN_TT_ConfigTypeDef * pTTParams)
```

Function description

Initialize TT operation parameters.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.
- **pTTParams:** pointer to a FDCAN_TT_ConfigTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigReferenceMessage

Function name

```
HAL_StatusTypeDef HAL_FDCAN_TT_ConfigReferenceMessage (FDCAN_HandleTypeDefDef * hfdcan,
uint32_t IdType, uint32_t Identifier, uint32_t Payload)
```

Function description

Configure the reference message.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.
- **IdType:** Identifier Type. This parameter can be a value of
 - FDCAN_id_type.
- **Identifier:** Reference Identifier. This parameter must be a number between:
 - 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
 - 0 and 0xFFFFFFFF, if IdType is FDCAN_EXTENDED_ID
- **Payload:** Enable or disable the additional payload. This parameter can be a value of
 - FDCAN_TT_Reference_Message_Payload. This parameter is ignored in case of time slaves. If this parameter is set to FDCAN_TT_REF_MESSAGE_ADD_PAYLOAD, the following elements are taken from Tx Buffer 0:
 - MessageMarker
 - TxEventFifoControl
 - DataLength
 - Data Bytes (payload):
 - bytes 2-8, for Level 1
 - bytes 5-8, for Level 0 and Level 2

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigTrigger

Function name

```
HAL_StatusTypeDef HAL_FDCAN_TT_ConfigTrigger (FDCAN_HandleTypeDefDef * hfdcan,
FDCAN_TriggerTypeDef * sTriggerConfig)
```

Function description

Configure the FDCAN trigger according to the specified parameters in the FDCAN_TriggerTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

- **sTriggerConfig:** pointer to an FDCAN_TriggerTypeDef structure that contains the trigger configuration information

Return values

- **HAL:** status

HAL_FDCAN_TT_SetGlobalTime

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetGlobalTime (FDCAN_HandleTypeDef * hfdcan, uint32_t TimePreset)

Function description

Schedule global time adjustment for the next reference message.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimePreset:** time preset value. This parameter must be a number between:
 - 0x0000 and 0xFFFF, Next_Master_Ref_Mark = Current_Master_Ref_Mark + TimePreset or:
 - 0x8001 and 0xFFFF, Next_Master_Ref_Mark = Current_Master_Ref_Mark - (0x10000 - TimePreset)

Return values

- **HAL:** status

HAL_FDCAN_TT_SetClockSynchronization

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetClockSynchronization (FDCAN_HandleTypeDef * hfdcan, uint32_t NewTURNumerator)

Function description

Schedule TUR numerator update for the next reference message.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **NewTURNumerator:** new value of the TUR numerator. This parameter must be a number between 0x10000 and 0x1FFF.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigStopWatch

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_ConfigStopWatch (FDCAN_HandleTypeDef * hfdcan, uint32_t Source, uint32_t Polarity)

Function description

Configure stop watch source and polarity.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **Source:** stop watch source. This parameter can be a value of
 - FDCAN_TT_stop_watch_source.

- **Polarity:** stop watch polarity. This parameter can be a value of
 - FDCAN_TT_stop_watch_polarity.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigRegisterTimeMark

Function name

```
HAL_StatusTypeDef HAL_FDCAN_TT_ConfigRegisterTimeMark (FDCAN_HandleTypeDef * hfdcan,  
          uint32_t TimeMarkSource, uint32_t TimeMarkValue, uint32_t RepeatFactor, uint32_t StartCycle)
```

Function description

Configure register time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimeMarkSource:** time mark source. This parameter can be a value of
 - FDCAN_TT_time_mark_source.
- **TimeMarkValue:** time mark value (reference). This parameter must be a number between 0 and 0xFFFF.
- **RepeatFactor:** repeat factor of the cycle for which the time mark is valid. This parameter can be a value of
 - FDCAN_TT_Repeat_Factor.
- **StartCycle:** index of the first cycle in which the time mark becomes valid. This parameter is ignored if RepeatFactor is set to FDCAN_TT_REPEAT_EVERY_CYCLE. This parameter must be a number between 0 and RepeatFactor.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableRegisterTimeMarkPulse

Function name

```
HAL_StatusTypeDef HAL_FDCAN_TT_EnableRegisterTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)
```

Function description

Enable register time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableRegisterTimeMarkPulse

Function name

```
HAL_StatusTypeDef HAL_FDCAN_TT_DisableRegisterTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)
```

Function description

Disable register time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableTriggerTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableTriggerTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable trigger time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableTriggerTimeMarkPulse

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableTriggerTimeMarkPulse (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable trigger time mark pulse generation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableHardwareGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableHardwareGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable gap control by input pin fdcan1_evt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableHardwareGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableHardwareGapControl (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable gap control by input pin fdcan1_evt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableTimeMarkGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableTimeMarkGapControl (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Enable gap control (finish only) by register time mark interrupt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableTimeMarkGapControl

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DisableTimeMarkGapControl (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Disable gap control by register time mark interrupt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_SetNextIsGap

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetNextIsGap (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Transmit next reference message with Next_is_Gap="1".

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_SetEndOfGap

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_SetEndOfGap (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Finish a Gap by requesting start of reference message.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigExternalSyncPhase

Function name

**HAL_StatusTypeDef HAL_FDCAN_TT_ConfigExternalSyncPhase (FDCAN_HandleTypeDefDef * hfdcan,
uint32_t TargetPhase)**

Function description

Configure target phase used for external synchronization by event trigger input pin fdcan1_evt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.
- **TargetPhase:** defines target value of cycle time when a rising edge of fdcan1_evt is expected. This parameter must be a number between 0 and 0xFFFF.

Return values

- **HAL:** status

HAL_FDCAN_TT_EnableExternalSynchronization

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_EnableExternalSynchronization (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Synchronize the phase of the FDCAN schedule to an external schedule using event trigger input pin fdcan1_evt.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_DisableExternalSynchronization

Function name

**HAL_StatusTypeDef HAL_FDCAN_TT_DisableExternalSynchronization (FDCAN_HandleTypeDefDef *
hfdcan)**

Function description

Disable external schedule synchronization.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TT_GetOperationStatus

Function name

**HAL_StatusTypeDef HAL_FDCAN_TT_GetOperationStatus (FDCAN_HandleTypeDef * hfdcan,
FDCAN_TTOperationStatusTypeDef * TTOpStatus)**

Function description

Get TT operation status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTOpStatus:** pointer to an FDCAN_TTOperationStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_ConfigInterruptLines

Function name

**HAL_StatusTypeDef HAL_FDCAN_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t
ITList, uint32_t InterruptLine)**

Function description

Assign interrupts to either Interrupt line 0 or 1.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ITList:** indicates which interrupts will be assigned to the selected interrupt line. This parameter can be any combination of
 - FDCAN_Interrupts.
- **InterruptLine:** Interrupt line. This parameter can be a value of
 - FDCAN_Interrupt_Line.

Return values

- **HAL:** status

HAL_FDCAN_TT_ConfigInterruptLines

Function name

**HAL_StatusTypeDef HAL_FDCAN_TT_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t
TTITList, uint32_t InterruptLine)**

Function description

Assign TT interrupts to either Interrupt line 0 or 1.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTITList:** indicates which interrupts will be assigned to the selected interrupt line. This parameter can be any combination of
 - FDCAN_TTIInterrupts.

- **InterruptLine:** Interrupt line. This parameter can be a value of
 - FDCAN_Interrupt_Line.

Return values

- **HAL:** status

HAL_FDCAN_ActivateNotification

Function name

```
HAL_StatusTypeDef HAL_FDCAN_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveITs, uint32_t BufferIndexes)
```

Function description

Enable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ActiveITs:** indicates which interrupts will be enabled. This parameter can be any combination of
 - FDCAN_Interrupts.
- **BufferIndexes:** Tx Buffer Indexes. This parameter can be any combination of
 - FDCAN_Tx_Location. This parameter is ignored if ActiveITs does not include one of the following:
 - FDCAN_IT_TX_COMPLETE
 - FDCAN_IT_TX_ABORT_COMPLETE

Return values

- **HAL:** status

HAL_FDCAN_DeactivateNotification

Function name

```
HAL_StatusTypeDef HAL_FDCAN_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveITs)
```

Function description

Disable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **InactiveITs:** indicates which interrupts will be disabled. This parameter can be any combination of
 - FDCAN_Interrupts.

Return values

- **HAL:** status

HAL_FDCAN_TT_ActivateNotification

Function name

```
HAL_StatusTypeDef HAL_FDCAN_TT_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveTTITs)
```

Function description

Enable TT interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ActiveTTITs:** indicates which TT interrupts will be enabled. This parameter can be any combination of
 - FDCAN_TTIInterrupts.

Return values

- **HAL:** status

HAL_FDCAN_TT_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_TT_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveTTITs)

Function description

Disable TT interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **InactiveTTITs:** indicates which TT interrupts will be disabled. This parameter can be any combination of
 - FDCAN_TTIInterrupts.

Return values

- **HAL:** status

HAL_FDCAN_IRQHandler

Function name

void HAL_FDCAN_IRQHandler (FDCAN_HandleTypeDef * hfdcan)

Function description

Handles FDCAN interrupt request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ClockCalibrationCallback

Function name

void HAL_FDCAN_ClockCalibrationCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t ClkCalibrationITs)

Function description

Clock Calibration callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ClkCalibrationITs:** indicates which Clock Calibration interrupts are signalled. This parameter can be any combination of

- FDCAN_Clock_Calibration_Interrupts.

Return values

- **None:**

HAL_FDCAN_TxEventFifoCallback

Function name

void HAL_FDCAN_TxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TxEventFifoITs)

Function description

Tx Event callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxEventFifoITs:** indicates which Tx Event FIFO interrupts are signalled. This parameter can be any combination of
 - FDCAN_Tx_Event_Fifo_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo0Callback

Function name

void HAL_FDCAN_RxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo0ITs)

Function description

Rx FIFO 0 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo0ITs:** indicates which Rx FIFO 0 interrupts are signalled. This parameter can be any combination of
 - FDCAN_Rx_Fifo0_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo1Callback

Function name

void HAL_FDCAN_RxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo1ITs)

Function description

Rx FIFO 1 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo1ITs:** indicates which Rx FIFO 1 interrupts are signalled. This parameter can be any combination of
 - FDCAN_Rx_Fifo1_Interrupts.

Return values

- **None:**

HAL_FDCAN_TxFifoEmptyCallback

Function name

`void HAL_FDCAN_TxFifoEmptyCallback (FDCAN_HandleTypeDef * hfdcan)`

Function description

Tx FIFO Empty callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TxBufferCompleteCallback

Function name

`void HAL_FDCAN_TxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)`

Function description

Transmission Complete callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the transmitted buffers. This parameter can be any combination of
 - FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_TxBufferAbortCallback

Function name

`void HAL_FDCAN_TxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)`

Function description

Transmission Cancellation callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the aborted buffers. This parameter can be any combination of
 - FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_RxBufferNewMessageCallback

Function name

`void HAL_FDCAN_RxBufferNewMessageCallback (FDCAN_HandleTypeDef * hfdcan)`

Function description

Rx Buffer New Message callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_HighPriorityMessageCallback

Function name

void HAL_FDCAN_HighPriorityMessageCallback (FDCAN_HandleTypeDefDef * hfdcan)

Function description

High Priority Message callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimestampWraparoundCallback

Function name

void HAL_FDCAN_TimestampWraparoundCallback (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Timestamp Wraparound callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimeoutOccurredCallback

Function name

void HAL_FDCAN_TimeoutOccurredCallback (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Timeout Occurred callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDefDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorCallback

Function name

void HAL_FDCAN_ErrorCallback (FDCAN_HandleTypeDefDef * hfdcan)

Function description

Error callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TT_ScheduleSyncCallback

Function name

```
void HAL_FDCAN_TT_ScheduleSyncCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t  
TTschedSyncITs)
```

Function description

TT Schedule Synchronization callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTschedSyncITs:** indicates which TT Schedule Synchronization interrupts are signalled. This parameter can be any combination of
 - FDCAN_TTScheduleSynchronization_Interrupts.

Return values

- **None:**

HAL_FDCAN_TT_TimeMarkCallback

Function name

```
void HAL_FDCAN_TT_TimeMarkCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TTTimeMarkITs)
```

Function description

TT Time Mark callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTTimeMarkITs:** indicates which TT Schedule Synchronization interrupts are signalled. This parameter can be any combination of
 - FDCAN_TTTimeMark_Interrupts.

Return values

- **None:**

HAL_FDCAN_TT_StopWatchCallback

Function name

```
void HAL_FDCAN_TT_StopWatchCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t SWTime, uint32_t  
SWCycleCount)
```

Function description

TT Stop Watch callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **SWTime:** Time Value captured at the Stop Watch Trigger pin (fdcan1_swt) falling/rising edge (as configured via HAL_FDCAN_TTCConfigStopWatch). This parameter is a number between 0 and 0xFFFF.
- **SWCycleCount:** Cycle count value captured together with SWTime. This parameter is a number between 0 and 0x3F.

Return values

- **None:**

HAL_FDCAN_TT_GlobalTimeCallback

Function name

void HAL_FDCAN_TT_GlobalTimeCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TTGlobTimelTs)

Function description

TT Global Time callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TTGlobTimelTs:** indicates which TT Global Time interrupts are signalled. This parameter can be any combination of
 - FDCAN_TTGlobalTime_Interrupts.

Return values

- **None:**

HAL_FDCAN_GetError

Function name

uint32_t HAL_FDCAN_GetError (FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN error code.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **FDCAN:** Error Code

HAL_FDCAN_GetState

Function name

HAL_FDCAN_StateTypeDef HAL_FDCAN_GetState (FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- HAL: state

23.3 FDCAN Firmware driver defines

23.3.1 FDCAN

FDCAN Bit Rate Switching

[FDCAN_BRS_OFF](#)

FDCAN frames transmitted/received without bit rate switching

[FDCAN_BRS_ON](#)

FDCAN frames transmitted/received with bit rate switching

FDCAN Calibration Counter

[FDCAN_CALIB_TIME_QUANTA_COUNTER](#)

Time Quanta Counter

[FDCAN_CALIB_CLOCK_PERIOD_COUNTER](#)

Oscillator Clock Period Counter

[FDCAN_CALIB_WATCHDOG_COUNTER](#)

Calibration Watchdog Counter

FDCAN Calibration Field Length

[FDCAN_CALIB_FIELD_LENGTH_32](#)

Calibration field length is 32 bits

[FDCAN_CALIB_FIELD_LENGTH_64](#)

Calibration field length is 64 bits

FDCAN Calibration State

[FDCAN_CLOCK_NOT_CALIBRATED](#)

Clock not calibrated

[FDCAN_CLOCK_BASIC_CALIBRATED](#)

Clock basic calibrated

[FDCAN_CLOCK_PRECISION_CALIBRATED](#)

Clock precision calibrated

Clock Calibration Interrupts

[FDCAN_IT_CALIB_STATE_CHANGED](#)

Clock calibration state changed

[FDCAN_IT_CALIB_WATCHDOG_EVENT](#)

Clock calibration watchdog event occurred

FDCAN Clock Divider

[FDCAN_CLOCK_DIV1](#)

Divide kernel clock by 1

FDCAN_CLOCK_DIV2

Divide kernel clock by 2

FDCAN_CLOCK_DIV4

Divide kernel clock by 4

FDCAN_CLOCK_DIV6

Divide kernel clock by 6

FDCAN_CLOCK_DIV8

Divide kernel clock by 8

FDCAN_CLOCK_DIV10

Divide kernel clock by 10

FDCAN_CLOCK_DIV12

Divide kernel clock by 12

FDCAN_CLOCK_DIV14

Divide kernel clock by 14

FDCAN_CLOCK_DIV16

Divide kernel clock by 16

FDCAN_CLOCK_DIV18

Divide kernel clock by 18

FDCAN_CLOCK_DIV20

Divide kernel clock by 20

FDCAN_CLOCK_DIV22

Divide kernel clock by 22

FDCAN_CLOCK_DIV24

Divide kernel clock by 24

FDCAN_CLOCK_DIV26

Divide kernel clock by 26

FDCAN_CLOCK_DIV28

Divide kernel clock by 28

FDCAN_CLOCK_DIV30

Divide kernel clock by 30

FDCAN communication state**FDCAN_COM_STATE_SYNC**

Node is synchronizing on CAN communication

FDCAN_COM_STATE_IDLE

Node is neither receiver nor transmitter

FDCAN_COM_STATE_RX

Node is operating as receiver

FDCAN_COM_STATE_TX

Node is operating as transmitter

FDCAN Counter Interrupts**FDCAN_IT_TIMESTAMP_WRAPAROUND**

Timestamp counter wrapped around

FDCAN_IT_TIMEOUT_OCCURRED

Timeout reached

FDCAN Data Field Size**FDCAN_DATA_BYTES_8**

8 bytes data field

FDCAN_DATA_BYTES_12

12 bytes data field

FDCAN_DATA_BYTES_16

16 bytes data field

FDCAN_DATA_BYTES_20

20 bytes data field

FDCAN_DATA_BYTES_24

24 bytes data field

FDCAN_DATA_BYTES_32

32 bytes data field

FDCAN_DATA_BYTES_48

48 bytes data field

FDCAN_DATA_BYTES_64

64 bytes data field

FDCAN Data Length Code**FDCAN_DLC_BYTES_0**

0 bytes data field

FDCAN_DLC_BYTES_1

1 bytes data field

FDCAN_DLC_BYTES_2

2 bytes data field

FDCAN_DLC_BYTES_3

3 bytes data field

FDCAN_DLC_BYTES_4

4 bytes data field

FDCAN_DLC_BYTES_5

5 bytes data field

FDCAN_DLC_BYTES_6

6 bytes data field

FDCAN_DLC_BYTES_7

7 bytes data field

FDCAN_DLC_BYTES_8

8 bytes data field

FDCAN_DLC_BYTES_12

12 bytes data field

FDCAN_DLC_BYTES_16

16 bytes data field

FDCAN_DLC_BYTES_20

20 bytes data field

FDCAN_DLC_BYTES_24

24 bytes data field

FDCAN_DLC_BYTES_32

32 bytes data field

FDCAN_DLC_BYTES_48

48 bytes data field

FDCAN_DLC_BYTES_64

64 bytes data field

FDCAN Error Interrupts**FDCAN_IT_RAM_ACCESS_FAILURE**

Message RAM access failure occurred

FDCAN_IT_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_IT_ERROR_PASSIVE

Error_Passive status changed

FDCAN_IT_ERROR_WARNING

Error_Warning status changed

FDCAN_IT_BUS_OFF

Bus_Off status changed

FDCAN_IT_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_IT_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_IT_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_IT_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN Error State Indicator

FDCAN_ESI_ACTIVE

Transmitting node is error active

FDCAN_ESI_PASSIVE

Transmitting node is error passive

FDCAN Event Type

FDCAN_TX_EVENT

Tx event

FDCAN_TX_IN_SPITE_OF_ABORT

Transmission in spite of cancellation

FDCAN Exported Macros

_HAL_FDCAN_RESET_HANDLE_STATE

Description:

- Reset FDCAN handle state.

Parameters:

- _HANDLE_: FDCAN handle.

Return value:

- None

_HAL_FDCAN_ENABLE_IT

Description:

- Enable the specified FDCAN interrupts.

Parameters:

- _HANDLE_: FDCAN handle.
- _INTERRUPT_: FDCAN interrupt. This parameter can be any combination of
 - FDCAN Interrupts

Return value:

- None

_HAL_FDCAN_DISABLE_IT

Description:

- Disable the specified FDCAN interrupts.

Parameters:

- _HANDLE_: FDCAN handle.
- _INTERRUPT_: FDCAN interrupt. This parameter can be any combination of
 - FDCAN Interrupts

Return value:

- None

_HAL_FDCAN_GET_IT

Description:

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

Parameters:

- `__HANDLE__`: FDCAN handle.
- `__INTERRUPT__`: FDCAN interrupt. This parameter can be one of
 - `FDCAN_Interrupts`

Return value:

- None

[`__HAL_FDCAN_CLEAR_IT`](#)**Description:**

- Clear the specified FDCAN interrupts.

Parameters:

- `__HANDLE__`: FDCAN handle.
- `__INTERRUPT__`: specifies the interrupts to clear. This parameter can be any combination of
 - `FDCAN_Interrupts`

Return value:

- None

[`__HAL_FDCAN_GET_FLAG`](#)**Description:**

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

Parameters:

- `__HANDLE__`: FDCAN handle.
- `__FLAG__`: FDCAN flag. This parameter can be one of
 - `FDCAN_flags`

Return value:

- None

[`__HAL_FDCAN_CLEAR_FLAG`](#)**Description:**

- Clear the specified FDCAN flags.

Parameters:

- `__HANDLE__`: FDCAN handle.
- `__FLAG__`: specifies the flags to clear. This parameter can be any combination of
 - `FDCAN_flags`

Return value:

- None

[`__HAL_FDCAN_GET_IT_SOURCE`](#)**Description:**

- Check if the specified FDCAN interrupt source is enabled or disabled.

Parameters:

- `__HANDLE__`: FDCAN handle.
- `__INTERRUPT__`: specifies the FDCAN interrupt source to check. This parameter can be a value of
 - `FDCAN_Interrupts`

Return value:

- None

[__HAL_FDCAN_TT_ENABLE_IT](#)

Description:

- Enable the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be any combination of
 - FDCAN_TTIInterrupts

Return value:

- None

[__HAL_FDCAN_TT_DISABLE_IT](#)

Description:

- Disable the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be any combination of
 - FDCAN_TTIInterrupts

Return value:

- None

[__HAL_FDCAN_TT_GET_IT](#)

Description:

- Check whether the specified FDCAN TT interrupt is set or not.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN TT interrupt. This parameter can be one of
 - FDCAN_TTIInterrupts

Return value:

- None

[__HAL_FDCAN_TT_CLEAR_IT](#)

Description:

- Clear the specified FDCAN TT interrupts.

Parameters:

- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the TT interrupts to clear. This parameter can be any combination of
 - FDCAN_TTIInterrupts

Return value:

- None

[__HAL_FDCAN_TT_GET_FLAG](#)

Description:

- Check whether the specified FDCAN TT flag is set or not.

Parameters:

- `__HANDLE__`: FDCAN handle.
- `__FLAG__`: FDCAN TT flag. This parameter can be one of
 - `FDCAN_TTflags`

Return value:

- None

_HAL_FDCAN_TT_CLEAR_FLAG**Description:**

- Clear the specified FDCAN TT flags.

Parameters:

- `__HANDLE__`: FDCAN handle.
- `__FLAG__`: specifies the TT flags to clear. This parameter can be any combination of
 - `FDCAN_TTflags`

Return value:

- None

_HAL_FDCAN_TT_GET_IT_SOURCE**Description:**

- Check if the specified FDCAN TT interrupt source is enabled or disabled.

Parameters:

- `__HANDLE__`: FDCAN handle.
- `__INTERRUPT__`: specifies the FDCAN TT interrupt source to check. This parameter can be a value of
 - `FDCAN_TTIInterrupts`

Return value:

- None

FDCAN FIFO watermark**FDCAN_CFG_TX_EVENT_FIFO**

Tx event FIFO

FDCAN_CFG_RX_FIFO0

Rx FIFO0

FDCAN_CFG_RX_FIFO1

Rx FIFO1

FDCAN Filter Configuration**FDCAN_FILTER_DISABLE**

Disable filter element

FDCAN_FILTER_TO_RXFIFO0

Store in Rx FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1

Store in Rx FIFO 1 if filter matches

FDCAN_FILTER_REJECT

Reject ID if filter matches

FDCAN_FILTER_HP

Set high priority if filter matches

FDCAN_FILTER_TO_RXFIFO0_HP

Set high priority and store in FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1_HP

Set high priority and store in FIFO 1 if filter matches

FDCAN_FILTER_TO_RXBUFFER

Store into Rx Buffer, configuration of FilterType ignored

FDCAN Filter Type**FDCAN_FILTER_RANGE**

Range filter from FilterID1 to FilterID2

FDCAN_FILTER_DUAL

Dual ID filter for FilterID1 or FilterID2

FDCAN_FILTER_MASK

Classic filter: FilterID1 = filter, FilterID2 = mask

FDCAN_FILTER_RANGE_NO_EIDM

Range filter from FilterID1 to FilterID2, EIDM mask not applied

FDCAN Flags**FDCAN_FLAG_TX_COMPLETE**

Transmission Completed

FDCAN_FLAG_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_FLAG_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN_FLAG_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN_FLAG_RX_BUFFER_NEW_MESSAGE

At least one received message stored into a Rx Buffer

FDCAN_FLAG_TX_EVT_FIFO_elt_LOST

Tx Event FIFO element lost

FDCAN_FLAG_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_FLAG_TX_EVT_FIFO_WATERMARK

Tx Event FIFO fill level reached watermark

FDCAN_FLAG_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN_FLAG_RX_FIFO0_MESSAGE_LOST

Rx FIFO 0 message lost

FDCAN_FLAG_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_FLAG_RX_FIFO0_WATERMARK

Rx FIFO 0 fill level reached watermark

FDCAN_FLAG_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN_FLAG_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_FLAG_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_FLAG_RX_FIFO1_WATERMARK

Rx FIFO 1 fill level reached watermark

FDCAN_FLAG_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN_FLAG_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_FLAG_ERROR_PASSIVE

Error_Passive status changed

FDCAN_FLAG_ERROR_WARNING

Error_Warning status changed

FDCAN_FLAG_BUS_OFF

Bus_Off status changed

FDCAN_FLAG_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_FLAG_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_FLAG_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_FLAG_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN_FLAG_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_FLAG_TIMEOUT_OCCURRED

Timeout reached

FDCAN_FLAG_CALIB_STATE_CHANGED

Clock calibration state changed

FDCAN_FLAG_CALIB_WATCHDOG_EVENT

Clock calibration watchdog event occurred

FDCAN format**FDCAN_CLASSIC_CAN**

Frame transmitted/received in Classic CAN format

FDCAN_FD_CAN

Frame transmitted/received in FDCAN format

FDCAN Frame Format**FDCAN_FRAME_CLASSIC**

Classic mode

FDCAN_FRAME_FD_NO_BRS

FD mode without BitRate Switshing

FDCAN_FRAME_FD_BRS

FD mode with BitRate Switshing

FDCAN Frame Type**FDCAN_DATA_FRAME**

Data frame

FDCAN_REMOTE_FRAME

Remote frame

FDCAN High Priority Message Storage**FDCAN_HP_STORAGE_NO_FIFO**

No FIFO selected

FDCAN_HP_STORAGE_MSG_LOST

FIFO message lost

FDCAN_HP_STORAGE_RXFIFO0

Message stored in FIFO 0

FDCAN_HP_STORAGE_RXFIFO1

Message stored in FIFO 1

FDCAN ID Type**FDCAN_STANDARD_ID**

Standard ID element

FDCAN_EXTENDED_ID

Extended ID element

FDCAN interrupt line**FDCAN_INTERRUPT_LINE0**

Interrupt Line 0

FDCAN_INTERRUPT_LINE1

Interrupt Line 1

FDCAN non-matching frames**FDCAN_ACCEPT_IN_RX_FIFO0**

Accept in Rx FIFO 0

FDCAN_ACCEPT_IN_RX_FIFO1

Accept in Rx FIFO 1

FDCAN_REJECT

Reject

FDCAN Operating Mode**FDCAN_MODE_NORMAL**

Normal mode

FDCAN_MODE_RESTRICTED_OPERATION

Restricted Operation mode

FDCAN_MODE_BUS_MONITORING

Bus Monitoring mode

FDCAN_MODE_INTERNAL_LOOPBACK

Internal LoopBack mode

FDCAN_MODE_EXTERNAL_LOOPBACK

External LoopBack mode

FDCAN Operation Mode**FDCAN_TT_COMMUNICATION_LEVEL1**

Time triggered communication, level 1

FDCAN_TT_COMMUNICATION_LEVEL2

Time triggered communication, level 2

FDCAN_TT_COMMUNICATION_LEVEL0

Time triggered communication, level 0

FDCAN protocol error code**FDCAN_PROTOCOL_ERROR_NONE**

No error occurred

FDCAN_PROTOCOL_ERROR_STUFF

Stuff error

FDCAN_PROTOCOL_ERROR_FORM

Form error

FDCAN_PROTOCOL_ERROR_ACK

Acknowledge error

FDCAN_PROTOCOL_ERROR_BIT1

Bit 1 (recessive) error

FDCAN_PROTOCOL_ERROR_BIT0

Bit 0 (dominant) error

FDCAN_PROTOCOL_ERROR_CRC

CRC check sum error

FDCAN_PROTOCOL_ERROR_NO_CHANGE

No change since last read

FDCAN Rx FIFO 0 Interrupts**FDCAN_IT_RX_FIFO0_MESSAGE_LOST**

Rx FIFO 0 message lost

FDCAN_IT_RX_FIFO0_FULL

Rx FIFO 0 full

FDCAN_IT_RX_FIFO0_WATERMARK

Rx FIFO 0 fill level reached watermark

FDCAN_IT_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN Rx FIFO 1 Interrupts**FDCAN_IT_RX_FIFO1_MESSAGE_LOST**

Rx FIFO 1 message lost

FDCAN_IT_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_IT_RX_FIFO1_WATERMARK

Rx FIFO 1 fill level reached watermark

FDCAN_IT_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN FIFO operation mode**FDCAN_RX_FIFO_BLOCKING**

Rx FIFO blocking mode

FDCAN_RX_FIFO_OVERWRITE

Rx FIFO overwrite mode

FDCAN Rx Interrupts**FDCAN_IT_RX_HIGH_PRIORITY_MSG**

High priority message received

FDCAN_IT_RX_BUFFER_NEW_MESSAGE

At least one received message stored into a Rx Buffer

FDCAN Rx Location**FDCAN_RX_FIFO0**

Get received message from Rx FIFO 0

FDCAN_RX_FIFO1

Get received message from Rx FIFO 1

FDCAN_RX_BUFFER0

Get received message from Rx Buffer 0

FDCAN_RX_BUFFER1

Get received message from Rx Buffer 1

FDCAN_RX_BUFFER2

Get received message from Rx Buffer 2

FDCAN_RX_BUFFER3

Get received message from Rx Buffer 3

FDCAN_RX_BUFFER4

Get received message from Rx Buffer 4

FDCAN_RX_BUFFER5

Get received message from Rx Buffer 5

FDCAN_RX_BUFFER6

Get received message from Rx Buffer 6

FDCAN_RX_BUFFER7

Get received message from Rx Buffer 7

FDCAN_RX_BUFFER8

Get received message from Rx Buffer 8

FDCAN_RX_BUFFER9

Get received message from Rx Buffer 9

FDCAN_RX_BUFFER10

Get received message from Rx Buffer 10

FDCAN_RX_BUFFER11

Get received message from Rx Buffer 11

FDCAN_RX_BUFFER12

Get received message from Rx Buffer 12

FDCAN_RX_BUFFER13

Get received message from Rx Buffer 13

FDCAN_RX_BUFFER14

Get received message from Rx Buffer 14

FDCAN_RX_BUFFER15

Get received message from Rx Buffer 15

FDCAN_RX_BUFFER16

Get received message from Rx Buffer 16

FDCAN_RX_BUFFER17

Get received message from Rx Buffer 17

FDCAN_RX_BUFFER18

Get received message from Rx Buffer 18

FDCAN_RX_BUFFER19

Get received message from Rx Buffer 19

FDCAN_RX_BUFFER20

Get received message from Rx Buffer 20

FDCAN_RX_BUFFER21

Get received message from Rx Buffer 21

FDCAN_RX_BUFFER22

Get received message from Rx Buffer 22

FDCAN_RX_BUFFER23

Get received message from Rx Buffer 23

FDCAN_RX_BUFFER24

Get received message from Rx Buffer 24

FDCAN_RX_BUFFER25

Get received message from Rx Buffer 25

FDCAN_RX_BUFFER26

Get received message from Rx Buffer 26

FDCAN_RX_BUFFER27

Get received message from Rx Buffer 27

FDCAN_RX_BUFFER28

Get received message from Rx Buffer 28

FDCAN_RX_BUFFER29

Get received message from Rx Buffer 29

FDCAN_RX_BUFFER30

Get received message from Rx Buffer 30

FDCAN_RX_BUFFER31

Get received message from Rx Buffer 31

FDCAN_RX_BUFFER32

Get received message from Rx Buffer 32

FDCAN_RX_BUFFER33

Get received message from Rx Buffer 33

FDCAN_RX_BUFFER34

Get received message from Rx Buffer 34

FDCAN_RX_BUFFER35

Get received message from Rx Buffer 35

FDCAN_RX_BUFFER36

Get received message from Rx Buffer 36

FDCAN_RX_BUFFER37

Get received message from Rx Buffer 37

FDCAN_RX_BUFFER38

Get received message from Rx Buffer 38

FDCAN_RX_BUFFER39

Get received message from Rx Buffer 39

FDCAN_RX_BUFFER40

Get received message from Rx Buffer 40

FDCAN_RX_BUFFER41

Get received message from Rx Buffer 41

FDCAN_RX_BUFFER42

Get received message from Rx Buffer 42

FDCAN_RX_BUFFER43

Get received message from Rx Buffer 43

FDCAN_RX_BUFFER44

Get received message from Rx Buffer 44

FDCAN_RX_BUFFER45

Get received message from Rx Buffer 45

FDCAN_RX_BUFFER46

Get received message from Rx Buffer 46

FDCAN_RX_BUFFER47

Get received message from Rx Buffer 47

FDCAN_RX_BUFFER48

Get received message from Rx Buffer 48

FDCAN_RX_BUFFER49

Get received message from Rx Buffer 49

FDCAN_RX_BUFFER50

Get received message from Rx Buffer 50

FDCAN_RX_BUFFER51

Get received message from Rx Buffer 51

FDCAN_RX_BUFFER52

Get received message from Rx Buffer 52

FDCAN_RX_BUFFER53

Get received message from Rx Buffer 53

FDCAN_RX_BUFFER54

Get received message from Rx Buffer 54

FDCAN_RX_BUFFER55

Get received message from Rx Buffer 55

FDCAN_RX_BUFFER56

Get received message from Rx Buffer 56

FDCAN_RX_BUFFER57

Get received message from Rx Buffer 57

FDCAN_RX_BUFFER58

Get received message from Rx Buffer 58

FDCAN_RX_BUFFER59

Get received message from Rx Buffer 59

FDCAN_RX_BUFFER60

Get received message from Rx Buffer 60

FDCAN_RX_BUFFER61

Get received message from Rx Buffer 61

FDCAN_RX_BUFFER62

Get received message from Rx Buffer 62

FDCAN_RX_BUFFER63

Get received message from Rx Buffer 63

FDCAN timeout operation**FDCAN_TIMEOUT_CONTINUOUS**

Timeout continuous operation

FDCAN_TIMEOUT_TX_EVENT_FIFO

Timeout controlled by Tx Event FIFO

FDCAN_TIMEOUT_RX_FIFO0

Timeout controlled by Rx FIFO 0

FDCAN_TIMEOUT_RX_FIFO1

Timeout controlled by Rx FIFO 1

FDCAN timestamp**FDCAN_TIMESTAMP_INTERNAL**

Timestamp counter value incremented according to TCP

FDCAN_TIMESTAMP_EXTERNAL

External timestamp counter value used

FDCAN timestamp prescaler**FDCAN_TIMESTAMP_PRESC_1**

Timestamp counter time unit in equal to CAN bit time

FDCAN_TIMESTAMP_PRESC_2

Timestamp counter time unit in equal to CAN bit time multiplied by 2

FDCAN_TIMESTAMP_PRESC_3

Timestamp counter time unit in equal to CAN bit time multiplied by 3

FDCAN_TIMESTAMP_PRESC_4

Timestamp counter time unit in equal to CAN bit time multiplied by 4

FDCAN_TIMESTAMP_PRESC_5

Timestamp counter time unit in equal to CAN bit time multiplied by 5

FDCAN_TIMESTAMP_PRESC_6

Timestamp counter time unit in equal to CAN bit time multiplied by 6

FDCAN_TIMESTAMP_PRESC_7

Timestamp counter time unit in equal to CAN bit time multiplied by 7

FDCAN_TIMESTAMP_PRESC_8

Timestamp counter time unit in equal to CAN bit time multiplied by 8

FDCAN_TIMESTAMP_PRESC_9

Timestamp counter time unit in equal to CAN bit time multiplied by 9

FDCAN_TIMESTAMP_PRESC_10

Timestamp counter time unit in equal to CAN bit time multiplied by 10

FDCAN_TIMESTAMP_PRESC_11

Timestamp counter time unit in equal to CAN bit time multiplied by 11

FDCAN_TIMESTAMP_PRESC_12

Timestamp counter time unit in equal to CAN bit time multiplied by 12

FDCAN_TIMESTAMP_PRESC_13

Timestamp counter time unit in equal to CAN bit time multiplied by 13

FDCAN_TIMESTAMP_PRESC_14

Timestamp counter time unit in equal to CAN bit time multiplied by 14

FDCAN_TIMESTAMP_PRESC_15

Timestamp counter time unit in equal to CAN bit time multiplied by 15

FDCAN_TIMESTAMP_PRESC_16

Timestamp counter time unit in equal to CAN bit time multiplied by 16

FDCAN TT Disturbing Error Interrupts**FDCAN_TT_IT_GLOBAL_TIME_ERROR**

Global Time Error

FDCAN_TT_IT_TX_COUNT_UNDERFLOW

Tx Count Underflow

FDCAN_TT_IT_TX_COUNT_OVERFLOW

Tx Count Overflow

FDCAN_TT_IT_SCHEDULING_ERROR_1

Scheduling Error 1

FDCAN_TT_IT_SCHEDULING_ERROR_2

Scheduling Error 2

FDCAN_TT_IT_ERROR_LEVEL_CHANGE

Error Level Changed

FDCAN TT Fatal Error Interrupts**FDCAN_TT_IT_INIT_WATCH_TRIGGER**

Initialization Watch Trigger

FDCAN_TT_IT_WATCH_TRIGGER

Watch Trigger

FDCAN_TT_IT_APPLICATION_WATCHDOG

Application Watchdog

FDCAN_TT_IT_CONFIG_ERROR

Configuration Error

FDCAN TT Flags**FDCAN_TT_FLAG_BASIC_CYCLE_START**

Start of Basic Cycle

FDCAN_TT_FLAG_MATRIX_CYCLE_START

Start of Matrix Cycle

FDCAN_TT_FLAG_SYNC_MODE_CHANGE

Change of Synchronization Mode

FDCAN_TT_FLAG_START_OF_GAP

Start of Gap

FDCAN_TT_FLAG_REG_TIME_MARK

Register Time Mark Interrupt

FDCAN_TT_FLAG_TRIG_TIME_MARK

Trigger Time Mark Event Internal

FDCAN_TT_FLAG_STOP_WATCH

Stop Watch Event

FDCAN_TT_FLAG_GLOBAL_TIME_WRAP

Global Time Wrap

FDCAN_TT_FLAG_GLOBAL_TIME_DISC

Global Time Discontinuity

FDCAN_TT_FLAG_GLOBAL_TIME_ERROR

Global Time Error

FDCAN_TT_FLAG_TX_COUNT_UNDERFLOW

Tx Count Underflow

FDCAN_TT_FLAG_TX_COUNT_OVERFLOW

Tx Count Overflow

FDCAN_TT_FLAG_SCHEDULING_ERROR_1

Scheduling Error 1

FDCAN_TT_FLAG_SCHEDULING_ERROR_2

Scheduling Error 2

FDCAN_TT_FLAG_ERROR_LEVEL_CHANGE

Error Level Changed

FDCAN_TT_FLAG_INIT_WATCH_TRIGGER

Initialization Watch Trigger

FDCAN_TT_FLAG_WATCH_TRIGGER

Watch Trigger

FDCAN_TT_FLAG_APPLICATION_WATCHDOG

Application Watchdog

FDCAN_TT_FLAG_CONFIG_ERROR

Configuration Error

FDCAN TT Global Time Interrupts**FDCAN_TT_IT_GLOBAL_TIME_WRAP**

Global Time Wrap

FDCAN_TT_IT_GLOBAL_TIME_DISC

Global Time Discontinuity

FDCAN TT Schedule Synchronization Interrupts**FDCAN_TT_IT_BASIC_CYCLE_START**

Start of Basic Cycle

FDCAN_TT_IT_MATRIX_CYCLE_START

Start of Matrix Cycle

FDCAN_TT_IT_SYNC_MODE_CHANGE

Change of Synchronization Mode

FDCAN_TT_IT_START_OF_GAP

Start of Gap

FDCAN TT Stop Watch Interrupt**FDCAN_TT_IT_STOP_WATCH**

Stop Watch Event

FDCAN TT Time Mark Interrupts**FDCAN_TT_IT_REG_TIME_MARK**

Register Time Mark Interrupt

FDCAN_TT_IT_TRIG_TIME_MARK

Trigger Time Mark Event Internal

FDCAN TT Automatic Clock Calibration**FDCAN_TT_AUTO_CLK_CALIB_DISABLE**

Automatic clock calibration in Level 0,2 disabled

FDCAN_TT_AUTO_CLK_CALIB_ENABLE

Automatic clock calibration in Level 0,2 enabled

FDCAN TT Basic Cycle Number**FDCAN_TT_CYCLES_PER_MATRIX_1**

1 Basic Cycle per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_2

2 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_4

4 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_8

8 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_16

16 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_32

32 Basic Cycles per Matrix

FDCAN_TT_CYCLES_PER_MATRIX_64

64 Basic Cycles per Matrix

FDCAN TT Cycle Start Sync**FDCAN_TT_NO_SYNC_PULSE**

No sync pulse

FDCAN_TT_SYNC_BASIC_CYCLE_START

Sync pulse at start of basic cycle

FDCAN_TT_SYNC_MATRIX_START

Sync pulse at start of matrix

FDCAN TT Error Level**FDCAN_TT_NO_ERROR**

Severity 0 - No Error

FDCAN_TT_WARNING

Severity 1 - Warning

FDCAN_TT_ERROR

Severity 2 - Error

FDCAN_TT_SEVERE_ERROR

Severity 3 - Severe Error

FDCAN TT Event Trigger Polarity**FDCAN_TT_EVT_TRIG_POL_RISING**

Rising edge trigger

FDCAN_TT_EVT_TRIG_POL_FALLING

Falling edge trigger

FDCAN TT Event Trigger Selection**FDCAN_TT_EVENT_TRIGGER_0**

TIM2 selected as event trigger

FDCAN_TT_EVENT_TRIGGER_1

TIM3 selected as event trigger

FDCAN_TT_EVENT_TRIGGER_2

ETH selected as event trigger

FDCAN_TT_EVENT_TRIGGER_3

HRTIM selected as event trigger

FDCAN TT External Clock Synchronization**FDCAN_TT_EXT_CLK_SYNC_DISABLE**

External clock synchronization in Level 0,2 disabled

FDCAN_TT_EXT_CLK_SYNC_ENABLE

External clock synchronization in Level 0,2 enabled

FDCAN TT Global Time Filtering**FDCAN_TT_GLOB_TIME_FILT_DISABLE**

Global time filtering in Level 0,2 disabled

FDCAN_TT_GLOB_TIME_FILT_ENABLE

Global time filtering in Level 0,2 enabled

FDCAN TT Master State**FDCAN_TT_MASTER_OFF**

Master_Off, no master properties relevant

FDCAN_TT_TIME_SLAVE

Operating as Time Slave

FDCAN_TT_BACKUP_TIME_MASTER

Operating as Backup Time Master

FDCAN_TT_CURRENT_TIME_MASTER

Operating as current Time Master

FDCAN TT Operation**FDCAN_STRICTLY_TT_OPERATION**

Strictly time-triggered operation

FDCAN_EXT_EVT_SYNC_TT_OPERATION

External event-synchronized time-triggered operation

FDCAN TT reference message payload**FDCAN_TT_REF_MESSAGE_NO_PAYLOAD**

Reference message has no additional payload

FDCAN_TT_REF_MESSAGE_ADD_PAYLOAD

Additional payload is taken from Tx Buffer 0

FDCAN TT repeat factor**FDCAN_TT_REPEAT_EVERY_CYCLE**

Trigger valid for all cycles

FDCAN_TT_REPEAT_EVERY_2ND_CYCLE

Trigger valid every 2nd cycle

FDCAN_TT_REPEAT_EVERY_4TH_CYCLE

Trigger valid every 4th cycle

FDCAN_TT_REPEAT_EVERY_8TH_CYCLE

Trigger valid every 8th cycle

FDCAN_TT_REPEAT_EVERY_16TH_CYCLE

Trigger valid every 16th cycle

FDCAN_TT_REPEAT_EVERY_32ND_CYCLE

Trigger valid every 32nd cycle

FDCAN_TT_REPEAT_EVERY_64TH_CYCLE

Trigger valid every 64th cycle

FDCAN TT Stop Watch Polarity**FDCAN_TT_STOP_WATCH_RISING**

Selected stop watch source is captured at rising edge of fdcan1_swt

FDCAN_TT_STOP_WATCH_FALLING

Selected stop watch source is captured at falling edge of fdcan1_swt

FDCAN TT Stop Watch Source**FDCAN_TT_STOP_WATCH_DISABLED**

Stop Watch disabled

FDCAN_TT_STOP_WATCH_CYCLE_TIME

Actual value of cycle time is copied to Capture Time register (TTCPT.SWV)

FDCAN_TT_STOP_WATCH_LOCAL_TIME

Actual value of local time is copied to Capture Time register (TTCPT.SWV)

FDCAN_TT_STOP_WATCH_GLOBAL_TIME

Actual value of global time is copied to Capture Time register (TTCPT.SWV)

FDCAN TT Stop Watch Trigger Selection**FDCAN_TT_STOP_WATCH_TRIGGER_0**

TIM2 selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_1

TIM3 selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_2

ETH selected as stop watch trigger

FDCAN_TT_STOP_WATCH_TRIGGER_3

HRTIM selected as stop watch trigger

FDCAN TT Synchronization State**FDCAN_TT_OUT_OF_SYNC**

Out of Synchronization

FDCAN_TT_SYNCHRONIZING

Synchronizing to communication

FDCAN_TT_IN_GAP

Schedule suspended by Gap

FDCAN_TT_IN_SCHEDULE

Synchronized to schedule

FDCAN TT time mark event external**FDCAN_TT_TM_NO_EXTERNAL_EVENT**

No action

FDCAN_TT_TM_GEN_EXTERNAL_EVENT

External event (pulse) is generated when trigger becomes active

FDCAN TT time mark event internal**FDCAN_TT_TM_NO_INTERNAL_EVENT**

No action

FDCAN_TT_TM_GEN_INTERNAL_EVENT

Internal event is generated when trigger becomes active

FDCAN TT Time Mark Source**FDCAN_TT_REG_TIMEMARK_DISABLED**

No Register Time Mark Interrupt generated

FDCAN_TT_REG_TIMEMARK_CYC_TIME

Register Time Mark Interrupt if Time Mark = cycle time

FDCAN_TT_REG_TIMEMARK_LOC_TIME

Register Time Mark Interrupt if Time Mark = local time

FDCAN_TT_REG_TIMEMARK_GLO_TIME

Register Time Mark Interrupt if Time Mark = global time

FDCAN TT Time Master**FDCAN_TT_SLAVE**

Time slave

FDCAN_TT_POTENTIAL_MASTER

Potential time master

FDCAN TT trigger type

FDCAN_TT_TX_REF_TRIGGER

Transmit reference message in strictly time-triggered operation

FDCAN_TT_TX_REF_TRIGGER_GAP

Transmit reference message in external event-synchronized time-triggered operation

FDCAN_TT_TX_TRIGGER_SINGLE

Start a single transmission in an exclusive time window

FDCAN_TT_TX_TRIGGER_CONTINUOUS

Start a continuous transmission in an exclusive time window

FDCAN_TT_TX_TRIGGER_ARBITRATION

Start a transmission in an arbitration time window

FDCAN_TT_TX_TRIGGER_MERGED

Start a merged arbitration window

FDCAN_TT_WATCH_TRIGGER

Check for missing reference messages in strictly time-triggered operation

FDCAN_TT_WATCH_TRIGGER_GAP

Check for missing reference messages in external event-synchronized time-triggered operation

FDCAN_TT_RX_TRIGGER

Check for the reception of periodic messages in exclusive time windows

FDCAN_TT_TIME_BASE_TRIGGER

Generate internal/external events depending on TmEventInt/TmEventExt configuration

FDCAN_TT_END_OF_LIST

Illegal trigger, to be assigned to the unused triggers after a FDCAN_TT_WATCH_TRIGGER or FDCAN_TT_WATCH_TRIGGER_GAP

FDCAN Tx FIFO/Queue Mode**FDCAN_TX_FIFO_OPERATION**

FIFO mode

FDCAN_TX_QUEUE_OPERATION

Queue mode

FDCAN Tx Event FIFO Interrupts**FDCAN_IT_TX_EVT_FIFO_elt_lost**

Tx Event FIFO element lost

FDCAN_IT_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_IT_TX_EVT_FIFO_WATERMARK

Tx Event FIFO fill level reached watermark

FDCAN_IT_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN Tx Interrupts

FDCAN_IT_TX_COMPLETE

Transmission Completed

FDCAN_IT_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_IT_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN Tx Location**FDCAN_TX_BUFFER0**

Add message to Tx Buffer 0

FDCAN_TX_BUFFER1

Add message to Tx Buffer 1

FDCAN_TX_BUFFER2

Add message to Tx Buffer 2

FDCAN_TX_BUFFER3

Add message to Tx Buffer 3

FDCAN_TX_BUFFER4

Add message to Tx Buffer 4

FDCAN_TX_BUFFER5

Add message to Tx Buffer 5

FDCAN_TX_BUFFER6

Add message to Tx Buffer 6

FDCAN_TX_BUFFER7

Add message to Tx Buffer 7

FDCAN_TX_BUFFER8

Add message to Tx Buffer 8

FDCAN_TX_BUFFER9

Add message to Tx Buffer 9

FDCAN_TX_BUFFER10

Add message to Tx Buffer 10

FDCAN_TX_BUFFER11

Add message to Tx Buffer 11

FDCAN_TX_BUFFER12

Add message to Tx Buffer 12

FDCAN_TX_BUFFER13

Add message to Tx Buffer 13

FDCAN_TX_BUFFER14

Add message to Tx Buffer 14

FDCAN_TX_BUFFER15

Add message to Tx Buffer 15

FDCAN_TX_BUFFER16

Add message to Tx Buffer 16

FDCAN_TX_BUFFER17

Add message to Tx Buffer 17

FDCAN_TX_BUFFER18

Add message to Tx Buffer 18

FDCAN_TX_BUFFER19

Add message to Tx Buffer 19

FDCAN_TX_BUFFER20

Add message to Tx Buffer 20

FDCAN_TX_BUFFER21

Add message to Tx Buffer 21

FDCAN_TX_BUFFER22

Add message to Tx Buffer 22

FDCAN_TX_BUFFER23

Add message to Tx Buffer 23

FDCAN_TX_BUFFER24

Add message to Tx Buffer 24

FDCAN_TX_BUFFER25

Add message to Tx Buffer 25

FDCAN_TX_BUFFER26

Add message to Tx Buffer 26

FDCAN_TX_BUFFER27

Add message to Tx Buffer 27

FDCAN_TX_BUFFER28

Add message to Tx Buffer 28

FDCAN_TX_BUFFER29

Add message to Tx Buffer 29

FDCAN_TX_BUFFER30

Add message to Tx Buffer 30

FDCAN_TX_BUFFER31

Add message to Tx Buffer 31

24 HAL FLASH Generic Driver

24.1 FLASH Firmware driver registers structures

24.1.1 **FLASH_ProcTypeDef**

Data Fields

- `__IO FLASH_ProcedureTypeDef ProcedureOnGoing`
- `__IO uint32_t NbSectorsToErase`
- `__IO uint8_t VoltageForErase`
- `__IO uint32_t Sector`
- `__IO uint32_t Address`
- `HAL_LockTypeDef Lock`
- `__IO uint32_t ErrorCode`

Field Documentation

- `__IO FLASH_ProcedureTypeDef FLASH_ProcTypeDef::ProcedureOnGoing`
Internal variable to indicate which procedure is ongoing or not in IT context
- `__IO uint32_t FLASH_ProcTypeDef::NbSectorsToErase`
Internal variable to save the remaining sectors to erase in IT context
- `__IO uint8_t FLASH_ProcTypeDef::VoltageForErase`
Internal variable to provide voltage range selected by user in IT context
- `__IO uint32_t FLASH_ProcTypeDef::Sector`
Internal variable to define the current sector which is erasing
- `__IO uint32_t FLASH_ProcTypeDef::Address`
Internal variable to save address selected for program
- `HAL_LockTypeDef FLASH_ProcTypeDef::Lock`
FLASH locking object
- `__IO uint32_t FLASH_ProcTypeDef::ErrorCode`
FLASH error code

24.2 FLASH Firmware driver API description

24.2.1 **FLASH peripheral features**

The Flash memory interface manages CPU AXI I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Option bytes programming
- Error code correction (ECC) : Data in flash are 266-bits word (10 bits added per double word)

24.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32H7xx devices.

1. FLASH Memory IO Programming functions:
 - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
 - Program functions: double word only
 - There Two modes of programming :
 - Polling mode using HAL_FLASH_Program() function
 - Interrupt mode using HAL_FLASH_Program_IT() function
2. Interrupts and flags management functions :
 - Handle FLASH interrupts by calling HAL_FLASH_IRQHandler()
 - Callback functions are called when the flash operations are finished :
HAL_FLASH_EndOfOperationCallback() when everything is ok, otherwise
HAL_FLASH_OperationErrorHandler()
 - Get error flag status by calling HAL_FLASH_GetError()
3. Option bytes management functions :
 - Lock and Unlock the option bytes using HAL_FLASH_OB_Unlock() and HAL_FLASH_OB_Lock() functions
 - Launch the reload of the option bytes using HAL_FLASH_Launch() function. In this case, a reset is generated

In addition to these functions, this driver includes a set of macros allowing to handle the following operations:

- Set the latency
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

Note: For any Flash memory program operation (erase or program), the CPU clock frequency (HCLK) must be at least 1MHz.

Note: The contents of the Flash memory are not guaranteed if a device reset occurs during a Flash memory operation.

Note: Any attempt to read the Flash memory while it is being written or erased, causes the bus to stall. Read operations are processed correctly once the program operation has completed. This means that code or data fetches cannot be performed while a write/erase operation is ongoing

24.2.3 Detailed description of functions

HAL_FLASH_Program

Function name

```
HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t DataAddress)
```

Function description

Program flash word of 256 bits at a specified address.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program
- **FlashAddress:** specifies the address to be programmed.
- **DataAddress:** specifies the address of data (256 bits) to be programmed

Return values

- **HAL_StatusTypeDef:** HAL Status

HAL_FLASH_Program_IT

Function name

```
HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t DataAddress)
```

Function description

Program flash words of 256 bits at a specified address with interrupt enabled.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program
- **FlashAddress:** specifies the address to be programmed.
- **DataAddress:** specifies the address of data (256 bits) to be programmed

Return values

- **HAL:** Status

HAL_FLASH_IRQHandler

Function name

```
void HAL_FLASH_IRQHandler (void )
```

Function description

This function handles FLASH interrupt request.

Return values

- **None:**

HAL_FLASH_EndOfOperationCallback

Function name

```
void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)
```

Function description

FLASH end of operation interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase Bank number which has been requested to erase Sectors Erase: Sector which has been erased (if 0xFFFFFFFF, it means that all the selected sectors have been erased) Program Address which was selected for data program

Return values

- **None:**

HAL_FLASH_OperationErrorCallback

Function name

```
void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)
```

Function description

FLASH operation error interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Sectors Erase: Sector number which returned an error Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_Unlock (void)

Function description

Unlock the FLASH control registers access.

Return values

- **HAL:** Status

HAL_FLASH_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_Lock (void)

Function description

Locks the FLASH control registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)

Function description

Unlock the FLASH Option Control Registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)

Function description

Lock the FLASH Option Control Registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Launch

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)

Function description

Launch the option byte loading.

Return values

- **HAL:** Status

HAL_FLASH_GetError

Function name

uint32_t HAL_FLASH_GetError (void)

Function description

Get the specific FLASH error flag.

Return values

- **HAL_FLASH_ERRORCode:** The returned value can be:
 - HAL_FLASH_ERROR_NONE: No error set
 - HAL_FLASH_ERROR_WRP_BANK1: Write Protection Error on Bank 1
 - HAL_FLASH_ERROR_PGS_BANK1 : Program Sequence Error on Bank 1
 - HAL_FLASH_ERROR_STRB_BANK1 : Strobe Error on Bank 1
 - HAL_FLASH_ERROR_INC_BANK1 : Inconsistency Error on Bank 1
 - HAL_FLASH_ERROR_OPE_BANK1 : Operation Error on Bank 1
 - HAL_FLASH_ERROR_RDP_BANK1 : Read Protection Error on Bank 1
 - HAL_FLASH_ERROR_RDS_BANK1 : Read Secured Error on Bank 1
 - HAL_FLASH_ERROR_SNECC_BANK1: SNECC Error on Bank 1
 - HAL_FLASH_ERROR_DBECC_BANK1: Double Detection ECC on Bank 1
 - HAL_FLASH_ERROR_WRP_BANK2 : Write Protection Error on Bank 2
 - HAL_FLASH_ERROR_PGS_BANK2 : Program Sequence Error on Bank 2
 - HAL_FLASH_ERROR_STRB_BANK2 : Strobe Error on Bank 2
 - HAL_FLASH_ERROR_INC_BANK2 : Inconsistency Error on Bank 2
 - HAL_FLASH_ERROR_OPE_BANK2 : Operation Error on Bank 2
 - HAL_FLASH_ERROR_RDP_BANK2 : Read Protection Error on Bank 2
 - HAL_FLASH_ERROR_RDS_BANK2 : Read Secured Error on Bank 2
 - HAL_FLASH_ERROR_SNECC_BANK2: SNECC Error on Bank 2
 - HAL_FLASH_ERROR_DBECC_BANK2: Double Detection ECC on Bank 2

FLASH_WaitForLastOperation

Function name

HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t Timeout, uint32_t Bank)

Function description

Wait for a FLASH operation to complete.

Parameters

- **Timeout:** maximum flash operation timeout
- **Bank:** flash FLASH_BANK_1 or FLASH_BANK_2

Return values

- **HAL_StatusTypeDef:** HAL Status

FLASH_OB_WaitForLastOperation

Function name

HAL_StatusTypeDef FLASH_OB_WaitForLastOperation (uint32_t Timeout)

Function description

Wait for a FLASH Option Bytes change operation to complete.

Parameters

- **Timeout:** maximum flash operation timeout

Return values

- **HAL_StatusTypeDef:** HAL Status

24.3 FLASH Firmware driver defines

24.3.1 FLASH

FLASH Address

IS_FLASH_PROGRAM_ADDRESS_BANK1

IS_FLASH_PROGRAM_ADDRESS_BANK2

IS_FLASH_PROGRAM_ADDRESS

IS_BOOT_ADDRESS

IS_FLASH_BANK

IS_FLASH_BANK_EXCLUSIVE

FLASH Error Code

HAL_FLASH_ERROR_NONE

No error

HAL_FLASH_ERROR_WRP

Write Protection Error

HAL_FLASH_ERROR_PGS

Program Sequence Error

HAL_FLASH_ERROR_STRB

Strobe Error

HAL_FLASH_ERROR_INC

Inconsistency Error

HAL_FLASH_ERROR_OPE

Operation Error

HAL_FLASH_ERROR_RDP

Read Protection Error

HAL_FLASH_ERROR_RDS

Read Secured Error

HAL_FLASH_ERROR_SNECC

Single Detection ECC

HAL_FLASH_ERROR_DBECC

Double Detection ECC

HAL_FLASH_ERROR_WRP_BANK1

Write Protection Error on Bank 1

HAL_FLASH_ERROR_PGS_BANK1

Program Sequence Error on Bank 1

HAL_FLASH_ERROR_STRB_BANK1

Strobe Error on Bank 1

HAL_FLASH_ERROR_INC_BANK1

Inconsistency Error on Bank 1

HAL_FLASH_ERROR_OPE_BANK1

Operation Error on Bank 1

HAL_FLASH_ERROR_RDP_BANK1

Read Protection Error on Bank 1

HAL_FLASH_ERROR_RDS_BANK1

Read Secured Error on Bank 1

HAL_FLASH_ERROR_SNECC_BANK1

Single Detection ECC on Bank 1

HAL_FLASH_ERROR_DBECC_BANK1

Double Detection ECC on Bank 1

HAL_FLASH_ERROR_WRP_BANK2

Write Protection Error on Bank 2

HAL_FLASH_ERROR_PGS_BANK2

Program Sequence Error on Bank 2

HAL_FLASH_ERROR_STRB_BANK2

Strobe Error on Bank 2

HAL_FLASH_ERROR_INC_BANK2

Inconsistency Error on Bank 2

HAL_FLASH_ERROR_OPE_BANK2

Operation Error on Bank 2

HAL_FLASH_ERROR_RDP_BANK2

Read Protection Error on Bank 2

HAL_FLASH_ERROR_RDS_BANK2

Read Secured Error on Bank 2

HAL_FLASH_ERROR_SNECC_BANK2

Single Detection ECC on Bank 2

HAL_FLASH_ERROR_DBECC_BANK2

Double Detection ECC on Bank 2

HAL_FLASH_ERROR_OB_CHANGE

Option Byte Change Error

FLASH Exported Macros**_HAL_FLASH_SET_LATENCY****Description:**

- Set the FLASH Latency.

Parameters:

- _LATENCY_: FLASH Latency The value of this parameter depend on device used within the same series

Return value:

- none

_HAL_FLASH_GET_LATENCY**Description:**

- Get the FLASH Latency.

Return value:

- FLASH: Latency The value of this parameter depend on device used within the same series

_HAL_FLASH_ENABLE_IT_BANK1**Description:**

- Enable the specified FLASH interrupt.

Parameters:

- _INTERRUPT_ : FLASH interrupt In case of Bank 1 This parameter can be any combination of the following values:
 - FLASH_IT_EOP_BANK1 : End of FLASH Bank 1 Operation Interrupt source
 - FLASH_IT_WRPERR_BANK1 : Write Protection Error on Bank 1 Interrupt source
 - FLASH_IT_PGSERR_BANK1 : Program Sequence Error on Bank 1 Interrupt source
 - FLASH_IT_STRBERR_BANK1 : Strobe Error on Bank 1 Interrupt source
 - FLASH_IT_INCERR_BANK1 : Inconsistency Error on Bank 1 Interrupt source
 - FLASH_IT_OPERR_BANK1 : Operation Error on Bank 1 Interrupt source
 - FLASH_IT_RDPERR_BANK1 : Read protection Error on Bank 1 Interrupt source
 - FLASH_IT_RDSERR_BANK1 : Read secure Error on Bank 1 Interrupt source
 - FLASH_IT_SNECCERR_BANK1 : Single ECC Error Correction on Bank 1 Interrupt source
 - FLASH_IT_DBECCERR_BANK1 : Double Detection ECC Error on Bank 1 Interrupt source
 - FLASH_IT_CRCEND_BANK1 : CRC End on Bank 1 Interrupt source
 - FLASH_IT_EOP_BANK2 : End of FLASH Bank 2 Operation Interrupt source
 - FLASH_IT_WRPERR_BANK2 : Write Protection Error on Bank 2 Interrupt source
 - FLASH_IT_PGSERR_BANK2 : Program Sequence Error on Bank 2 Interrupt source
 - FLASH_IT_STRBERR_BANK2 : Strobe Error on Bank 2 Interrupt source
 - FLASH_IT_INCERR_BANK2 : Inconsistency Error on Bank 2 Interrupt source
 - FLASH_IT_OPERR_BANK2 : Operation Error on Bank 2 Interrupt source

- FLASH_IT_RDPERR_BANK2 : Read protection Error on Bank 2 Interrupt source
- FLASH_IT_RDSERR_BANK2 : Read secure Error on Bank 2 Interrupt source
- FLASH_IT_SNECCERR_BANK2 : Single ECC Error Correction on Bank 2 Interrupt source
- FLASH_IT_DBECCERR_BANK2 : Double Detection ECC Error on Bank 2 Interrupt source
- FLASH_IT_CRCEND_BANK2 : CRC End on Bank 2 Interrupt source

Description:

- none

[_HAL_FLASH_ENABLE_IT_BANK2](#)

[_HAL_FLASH_ENABLE_IT](#)

[_HAL_FLASH_DISABLE_IT_BANK1](#)

Description:

- Disable the specified FLASH interrupt.

Parameters:

- _INTERRUPT_ : FLASH interrupt In case of Bank 1 This parameter can be any combination of the following values:
 - FLASH_IT_EOP_BANK1 : End of FLASH Bank 1 Operation Interrupt source
 - FLASH_IT_WRPERR_BANK1 : Write Protection Error on Bank 1 Interrupt source
 - FLASH_IT_PGSERR_BANK1 : Program Sequence Error on Bank 1 Interrupt source
 - FLASH_IT_STRBERR_BANK1 : Strobe Error on Bank 1 Interrupt source
 - FLASH_IT_INCERR_BANK1 : Inconsistency Error on Bank 1 Interrupt source
 - FLASH_IT_OPERR_BANK1 : Operation Error on Bank 1 Interrupt source
 - FLASH_IT_RDPERR_BANK1 : Read protection Error on Bank 1 Interrupt source
 - FLASH_IT_RDSERR_BANK1 : Read secure Error on Bank 1 Interrupt source
 - FLASH_IT_SNECCERR_BANK1 : Single ECC Error Correction on Bank 1 Interrupt source
 - FLASH_IT_DBECCERR_BANK1 : Double Detection ECC Error on Bank 1 Interrupt source
 - FLASH_IT_CRCEND_BANK1 : CRC End on Bank 1 Interrupt source
 - FLASH_IT_EOP_BANK2 : End of FLASH Bank 2 Operation Interrupt source
 - FLASH_IT_WRPERR_BANK2 : Write Protection Error on Bank 2 Interrupt source
 - FLASH_IT_PGSERR_BANK2 : Program Sequence Error on Bank 2 Interrupt source
 - FLASH_IT_STRBERR_BANK2 : Strobe Error on Bank 2 Interrupt source
 - FLASH_IT_INCERR_BANK2 : Inconsistency Error on Bank 2 Interrupt source
 - FLASH_IT_OPERR_BANK2 : Operation Error on Bank 2 Interrupt source
 - FLASH_IT_RDPERR_BANK2 : Read protection Error on Bank 2 Interrupt source
 - FLASH_IT_RDSERR_BANK2 : Read secure Error on Bank 2 Interrupt source
 - FLASH_IT_SNECCERR_BANK2 : Single ECC Error Correction on Bank 2 Interrupt source
 - FLASH_IT_DBECCERR_BANK2 : Double Detection ECC Error on Bank 2 Interrupt source
 - FLASH_IT_CRCEND_BANK2 : CRC End on Bank 2 Interrupt source

Description:

- none

[_HAL_FLASH_DISABLE_IT_BANK2](#)

[_HAL_FLASH_DISABLE_IT](#)

[_HAL_FLASH_GET_FLAG_BANK1](#)

Description:

- Checks whether the specified FLASH flag is set or not.

Parameters:

- __FLAG__: specifies the FLASH flag to check. In case of Bank 1 This parameter can be any combination of the following values :
 - FLASH_FLAG_BSY_BANK1 : FLASH Bank 1 Busy flag
 - FLASH_FLAG_WBNE_BANK1 : Waiting for Data to Write on Bank 1 flag
 - FLASH_FLAG_QW_BANK1 : Write Waiting in Operation Queue on Bank 1 flag
 - FLASH_FLAG_CRC_BUSY_BANK1 : CRC module is working on Bank 1 flag
 - FLASH_FLAG_EOP_BANK1 : End Of Program on Bank 1 flag
 - FLASH_FLAG_WRPERR_BANK1 : Write Protection Error on Bank 1 flag
 - FLASH_FLAG_PGSERR_BANK1 : Program Sequence Error on Bank 1 flag
 - FLASH_FLAG_STRBER_BANK1 : Program Alignment Error on Bank 1 flag
 - FLASH_FLAG_INCERR_BANK1 : Inconsistency Error on Bank 1 flag
 - FLASH_FLAG_OPERR_BANK1 : Operation Error on Bank 1 flag
 - FLASH_FLAG_RDPERR_BANK1 : Read Protection Error on Bank 1 flag
 - FLASH_FLAG_RDSERR_BANK1 : Read secure Error on Bank 1 flag
 - FLASH_FLAG_SNECCE_BANK1 : Single ECC Error Correction on Bank 1 flag
 - FLASH_FLAG_DBECCE_BANK1 : Double Detection ECC Error on Bank 1 flag
 - FLASH_FLAG_CRCEND_BANK1 : CRC End on Bank 1 flag
 - FLASH_FLAG_BSY_BANK2 : FLASH Bank 2 Busy flag
 - FLASH_FLAG_WBNE_BANK2 : Waiting for Data to Write on Bank 2 flag
 - FLASH_FLAG_QW_BANK2 : Write Waiting in Operation Queue on Bank 2 flag
 - FLASH_FLAG_CRC_BUSY_BANK2 : CRC module is working on Bank 2 flag
 - FLASH_FLAG_EOP_BANK2 : End Of Program on Bank 2 flag
 - FLASH_FLAG_WRPERR_BANK2 : Write Protection Error on Bank 2 flag
 - FLASH_FLAG_PGSERR_BANK2 : Program Sequence Error on Bank 2 flag
 - FLASH_FLAG_STRBER_BANK2 : Program Alignment Error on Bank 2 flag
 - FLASH_FLAG_INCERR_BANK2 : Inconsistency Error on Bank 2 flag
 - FLASH_FLAG_OPERR_BANK2 : Operation Error on Bank 2 flag
 - FLASH_FLAG_RDPERR_BANK2 : Read Protection Error on Bank 2 flag
 - FLASH_FLAG_RDSERR_BANK2 : Read secure Error on Bank 2 flag
 - FLASH_FLAG_SNECCE_BANK2 : Single ECC Error Correction on Bank 2 flag
 - FLASH_FLAG_DBECCE_BANK2 : Double Detection ECC Error on Bank 2 flag
 - FLASH_FLAG_CRCEND_BANK2 : CRC End on Bank 2 flag

Return value:

- The: new state of FLASH_FLAG (SET or RESET).

[_HAL_FLASH_GET_FLAG_BANK2](#)

[_HAL_FLASH_GET_FLAG](#)

[_HAL_FLASH_CLEAR_FLAG_BANK1](#)

Description:

- Clear the specified FLASH flag.

Parameters:

- __FLAG__: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
 - FLASH_FLAG_EOP_BANK1 : End Of Program on Bank 1 flag
 - FLASH_FLAG_WRPERR_BANK1 : Write Protection Error on Bank 1 flag

- FLASH_FLAG_PGSERR_BANK1 : Program Sequence Error on Bank 1 flag
- FLASH_FLAG_STRBER_BANK1 : Program Alignment Error on Bank 1 flag
- FLASH_FLAG_INCERR_BANK1 : Inconsistency Error on Bank 1 flag
- FLASH_FLAG_OPERR_BANK1 : Operation Error on Bank 1 flag
- FLASH_FLAG_RDPERR_BANK1 : Read Protection Error on Bank 1 flag
- FLASH_FLAG_RDSERR_BANK1 : Read secure Error on Bank 1 flag
- FLASH_FLAG_SNECCE_BANK1 : Single ECC Error Correction on Bank 1 flag
- FLASH_FLAG_DBECCE_BANK1 : Double Detection ECC Error on Bank 1 flag
- FLASH_FLAG_CRCEND_BANK1 : CRC End on Bank 1 flag
- FLASH_FLAG_EOP_BANK2 : End Of Program on Bank 2 flag
- FLASH_FLAG_WRPERR_BANK2 : Write Protection Error on Bank 2 flag
- FLASH_FLAG_PGSERR_BANK2 : Program Sequence Error on Bank 2 flag
- FLASH_FLAG_STRBER_BANK2 : Program Alignment Error on Bank 2 flag
- FLASH_FLAG_INCERR_BANK2 : Inconsistency Error on Bank 2 flag
- FLASH_FLAG_OPERR_BANK2 : Operation Error on Bank 2 flag
- FLASH_FLAG_RDPERR_BANK2 : Read Protection Error on Bank 2 flag
- FLASH_FLAG_RDSERR_BANK2 : Read secure Error on Bank 2 flag
- FLASH_FLAG_SNECCE_BANK2 : Single ECC Error Correction on Bank 2 flag
- FLASH_FLAG_DBECCE_BANK2 : Double Detection ECC Error on Bank 2 flag
- FLASH_FLAG_CRCEND_BANK2 : CRC End on Bank 2 flag

Return value:

- none

[_HAL_FLASH_CLEAR_FLAG_BANK2](#)

[_HAL_FLASH_CLEAR_FLAG](#)

[_HAL_FLASH_CALC_BOOT_BASE_ADR](#)

Description:

- Calculate the FLASH Boot Base Adress (BOOT_ADD0 or BOOT_ADD1)

Parameters:

- __ADDRESS__: FLASH Boot Address (in the range 0x0000 0000 to 0x2004 FFFF with a granularity of 16KB)

Return value:

- The: FLASH Boot Base Adress

Notes:

- Returned value BOOT_ADDx[15:0] corresponds to boot address [29:14].

FLASH Flag definition

[FLASH_FLAG_BSY](#)

FLASH Busy flag

[FLASH_FLAG_WDW](#)

Waiting for Data to Write on flag

[FLASH_FLAG_QW](#)

Write Waiting in Operation Queue on flag

FLASH_FLAG_CRC_BUSY

CRC module is working on flag

FLASH_FLAG_EOP

End Of Program on flag

FLASH_FLAG_WRPERR

Write Protection Error on flag

FLASH_FLAG_PGSERR

Program Sequence Error on flag

FLASH_FLAG_STRBERR

strobe Error on flag

FLASH_FLAG_INCERR

Inconsistency Error on flag

FLASH_FLAG_OPERR

Operation Error on flag

FLASH_FLAG_RDPERR

Read Protection Error on flag

FLASH_FLAG_RDSERR

Read Secured Error on flag

FLASH_FLAG_SNECCERR

Single ECC Error Correction on flag

FLASH_FLAG_DBECCERR

Double Detection ECC Error on flag

FLASH_FLAG_CRCEND

CRC module completes on bank flag

FLASH_FLAG_BSY_BANK1

FLASH Bank 1 Busy flag

FLASH_FLAG_WBNE_BANK1

Waiting for Data to Write on Bank 1 flag

FLASH_FLAG_QW_BANK1

Write Waiting in Operation Queue on Bank 1 flag

FLASH_FLAG_CRC_BUSY_BANK1

CRC module is working on Bank 1 flag

FLASH_FLAG_EOP_BANK1

End Of Program on Bank 1 flag

FLASH_FLAG_WRPERR_BANK1

Write Protection Error on Bank 1 flag

FLASH_FLAG_PGSERR_BANK1

Program Sequence Error on Bank 1 flag

FLASH_FLAG_STRBER_BANK1R

strobe Error on Bank 1 flag

FLASH_FLAG_INCERR_BANK1

Inconsistency Error on Bank 1 flag

FLASH_FLAG_OPERR_BANK1

Operation Error on Bank 1 flag

FLASH_FLAG_RDPERR_BANK1

Read Protection Error on Bank 1 flag

FLASH_FLAG_RDSERR_BANK1

Read Secured Error on Bank 1 flag

FLASH_FLAG_SNECCE_BANK1RR

Single ECC Error Correction on Bank 1 flag

FLASH_FLAG_DBECCE_BANK1RR

Double Detection ECC Error on Bank 1 flag

FLASH_FLAG_CRCEND_BANK1

CRC module completes on bank Bank 1 flag

FLASH_FLAG_ALL_ERRORS_BANK1**FLASH_FLAG_ALL_BANK1****FLASH_FLAG_BSY_BANK2**

FLASH Bank 2 Busy flag

FLASH_FLAG_WBNE_BANK2

Waiting for Data to Write on Bank 2 flag

FLASH_FLAG_QW_BANK2

Write Waiting in Operation Queue on Bank 2 flag

FLASH_FLAG_CRC_BUSY_BANK2

CRC module is working on Bank 2 flag

FLASH_FLAG_EOP_BANK2

End Of Program on Bank 2 flag

FLASH_FLAG_WRPERR_BANK2

Write Protection Error on Bank 2 flag

FLASH_FLAG_PGSERR_BANK2

Program Sequence Error on Bank 2 flag

FLASH_FLAG_STRBER_BANK2R

Strobe Error on Bank 2 flag

FLASH_FLAG_INCERR_BANK2

Inconsistency Error on Bank 2 flag

FLASH_FLAG_OPERR_BANK2

Operation Error on Bank 2 flag

FLASH_FLAG_RDPERR_BANK2

Read Protection Error on Bank 2 flag

FLASH_FLAG_RDSERR_BANK2

Read Secured Error on Bank 2 flag

FLASH_FLAG_SNECCE_BANK2RR

Single ECC Error Correction on Bank 2 flag

FLASH_FLAG_DBECCE_BANK2RR

Double Detection ECC Error on Bank 2 flag

FLASH_FLAG_CRCEND_BANK2

CRC module completes on bank Bank 2 flag

FLASH_FLAG_ALL_ERRORS_BANK2**FLASH_FLAG_ALL_BANK2*****FLASH Interrupt definition*****FLASH_IT_EOP_BANK1**

End of FLASH Bank 1 Operation Interrupt source

FLASH_IT_WRPERR_BANK1

Write Protection Error on Bank 1 Interrupt source

FLASH_IT_PGSERR_BANK1

Program Sequence Error on Bank 1 Interrupt source

FLASH_IT_STRBERR_BANK1

Strobe Error on Bank 1 Interrupt source

FLASH_IT_INCERR_BANK1

Inconsistency Error on Bank 1 Interrupt source

FLASH_IT_OPERR_BANK1

Operation Error on Bank 1 Interrupt source

FLASH_IT_RDPERR_BANK1

Read protection Error on Bank 1 Interrupt source

FLASH_IT_RDSERR_BANK1

Read Secured Error on Bank 1 Interrupt source

FLASH_IT_SNECCERR_BANK1

Single ECC Error Correction on Bank 1 Interrupt source

FLASH_IT_DBECERR_BANK1

Double Detection ECC Error on Bank 1 Interrupt source

FLASH_IT_CRCEND_BANK1

CRC End on Bank 1 Interrupt source

FLASH_IT_ALL_BANK1

FLASH_IT_EOP_BANK2

End of FLASH Bank 2 Operation Interrupt source

FLASH_IT_WRPERR_BANK2

Write Protection Error on Bank 2 Interrupt source

FLASH_IT_PGSERR_BANK2

Program Sequence Error on Bank 2 Interrupt source

FLASH_IT_STRBERR_BANK2

Strobe Error on Bank 2 Interrupt source

FLASH_IT_INCERR_BANK2

Inconsistency Error on Bank 2 Interrupt source

FLASH_IT_OPERR_BANK2

Operation Error on Bank 2 Interrupt source

FLASH_IT_RDPERR_BANK2

Read protection Error on Bank 2 Interrupt source

FLASH_IT_RDSERR_BANK2

Read Secured Error on Bank 2 Interrupt source

FLASH_IT_SNECCERR_BANK2

Single ECC Error Correction on Bank 2 Interrupt source

FLASH_IT_DBECERR_BANK2

Double Detection ECC Error on Bank 2 Interrupt source

FLASH_IT_CRCEND_BANK2

CRC End on Bank 2 Interrupt source

FLASH_IT_ALL_BANK2

FLASH BANK IT Definitions

IS_FLASH_IT_BANK1

IS_FLASH_IT_BANK2

FLASH Definitions

IS_FLASH_TYPEPROGRAM

FLASH Keys

FLASH_KEY1

FLASH_KEY2

FLASH_OPT_KEY1

FLASH_OPT_KEY2

FLASH Latency

FLASH_LATENCY_0

FLASH Zero Latency cycle

FLASH_LATENCY_1

FLASH One Latency cycle

FLASH_LATENCY_2

FLASH Two Latency cycles

FLASH_LATENCY_3

FLASH Three Latency cycles

FLASH_LATENCY_4

FLASH Four Latency cycles

FLASH_LATENCY_5

FLASH Five Latency cycles

FLASH_LATENCY_6

FLASH Six Latency cycles

FLASH_LATENCY_7

FLASH Seven Latency cycles

FLASH Program Parallelism**FLASH_PSIZE_BYTE****FLASH_PSIZE_HALF_WORD****FLASH_PSIZE_WORD****FLASH_PSIZE_DOUBLE_WORD****CR_PSIZE_MASK*****FLASH Sectors*****FLASH_SECTOR_0**

Sector Number 0

FLASH_SECTOR_1

Sector Number 1

FLASH_SECTOR_2

Sector Number 2

FLASH_SECTOR_3

Sector Number 3

FLASH_SECTOR_4

Sector Number 4

FLASH_SECTOR_5

Sector Number 5

FLASH_SECTOR_6

Sector Number 6

FLASH_SECTOR_7

Sector Number 7

FLASH Type Program**FLASH_TYPEPROGRAM_FLASHWORD**

Program a flash word (256-bit) at a specified address

25 HAL FLASH Extension Driver

25.1 FLASHEx Firmware driver registers structures

25.1.1 FLASH_EraselInitTypeDef

Data Fields

- *uint32_t TypeErase*
- *uint32_t Banks*
- *uint32_t Sector*
- *uint32_t NbSectors*
- *uint32_t VoltageRange*

Field Documentation

- *uint32_t FLASH_EraselInitTypeDef::TypeErase*

Mass erase or sector Erase. This parameter can be a value of **FLASH Type Erase**

- *uint32_t FLASH_EraselInitTypeDef::Banks*

Select banks to erase when Mass erase is enabled. This parameter must be a value of **FLASH Banks**

- *uint32_t FLASH_EraselInitTypeDef::Sector*

Initial FLASH sector to erase when Mass erase is disabled This parameter must be a value of **FLASH Sectors**

- *uint32_t FLASH_EraselInitTypeDef::NbSectors*

Number of sectors to be erased. This parameter must be a value between 1 and (max number of sectors - value of Initial sector)

- *uint32_t FLASH_EraselInitTypeDef::VoltageRange*

The device voltage range which defines the erase parallelism This parameter must be a value of **FLASH Voltage Range**

25.1.2 FLASH_OBProgramInitTypeDef

Data Fields

- *uint32_t OptionType*
- *uint32_t WRPState*
- *uint32_t WRPSector*
- *uint32_t RDPLevel*
- *uint32_t BORLevel*
- *uint32_t USERType*
- *uint32_t USERConfig*
- *uint32_t Banks*
- *uint32_t PCROPConfig*
- *uint32_t PCROPStartAddr*
- *uint32_t PCROPEndAddr*
- *uint32_t BootConfig*
- *uint32_t BootAddr0*
- *uint32_t BootAddr1*
- *uint32_t SecureAreaConfig*

- `uint32_t SecureAreaStartAddr`
- `uint32_t SecureAreaEndAddr`

Field Documentation

- `uint32_t FLASH_OBProgramInitTypeDef::OptionType`

Option byte to be configured. This parameter can be a value of **FLASH Option Type**

- `uint32_t FLASH_OBProgramInitTypeDef::WRPState`

Write protection activation or deactivation. This parameter can be a value of **FLASH WRP State**

- `uint32_t FLASH_OBProgramInitTypeDef::WRPSector`

Specifies the sector(s) to be write protected. The value of this parameter depend on device used within the same series

- `uint32_t FLASH_OBProgramInitTypeDef::RDPLevel`

Set the read protection level. This parameter can be a value of **FLASH Option Bytes Read Protection**

- `uint32_t FLASH_OBProgramInitTypeDef::BORLevel`

Set the BOR Level. This parameter can be a value of **FLASH BOR Reset Level**

- `uint32_t FLASH_OBProgramInitTypeDef::USERType`

User option byte(s) to be configured (used for OPTIONBYTE_USER). This parameter can be a combination of **FLASHEx OB USER Type**

- `uint32_t FLASH_OBProgramInitTypeDef::USERConfig`

Program the FLASH User Option Byte: WWDG_SW / IWDG_SW / RST_STOP / RST_STDBY / IWDG_FREEZE_STOP / IWDG_FREEZE_SANDBY.

- `uint32_t FLASH_OBProgramInitTypeDef::Banks`

Select banks for WRP , PCROP and secure area config . This parameter must be a value of **FLASH Banks**

- `uint32_t FLASH_OBProgramInitTypeDef::PCROPConfig`

specifies if the PCROP area shall be erased or not when RDP level decreased from Level 1 to Level 0 or during a mass erase. This parameter must be a value of **FLASHEx OB PCROP RDP** enumeration

- `uint32_t FLASH_OBProgramInitTypeDef::PCROPStartAddr`

PCROP Start address (used for OPTIONBYTE_PCROP). This parameter must be a value between begin and end of a bank

- `uint32_t FLASH_OBProgramInitTypeDef::PCROPEndAddr`

PCROP End address (used for OPTIONBYTE_PCROP). This parameter must be a value between PCROP Start address and end of a bank

- `uint32_t FLASH_OBProgramInitTypeDef::BootConfig`

Specifies if the Boot Address to be configured BOOT_ADD0, BOOT_ADD1 or both. This parameter must be a value of **FLASHEx OB BOOT OPTION** enumeration

- `uint32_t FLASH_OBProgramInitTypeDef::BootAddr0`

Boot Address 0. This parameter must be a value between begin and end of a bank

- `uint32_t FLASH_OBProgramInitTypeDef::BootAddr1`

Boot Address 1. This parameter must be a value between begin and end of a bank

- `uint32_t FLASH_OBProgramInitTypeDef::SecureAreaConfig`

specifies if the bank secured area shall be erased or not when RDP level decreased from Level 1 to Level 0 or during a mass erase. This parameter must be a value of **FLASHEx OB SECURE RDP** enumeration

- `uint32_t FLASH_OBProgramInitTypeDef::SecureAreaStartAddr`

Bank Secure area Start address. This parameter must be a value between begin and end of bank1

- `uint32_t FLASH_OBProgramInitTypeDef::SecureAreaEndAddr`
Bank Secure area End address . This parameter must be a value between Start address and end of a bank1

25.2 FLASHEx Firmware driver API description

25.2.1 Flash Extension features

Comparing to other previous devices, the FLASH interface for STM32H7xx devices contains the following additional features

- Capacity up to 2 Mbyte with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank memory organization
- PCROP protection for all banks

25.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32H7xx devices. It includes

1. FLASH Memory Erase functions:
 - Lock and Unlock the FLASH interface using `HAL_FLASH_Unlock()` and `HAL_FLASH_Lock()` functions
 - Erase function: Erase sector, erase all sectors
 - There are two modes of erase :
 - Polling Mode using `HAL_FLASHEx_Erase()`
 - Interrupt Mode using `HAL_FLASHEx_Erase_IT()`
2. Option Bytes Programming functions: Use `HAL_FLASHEx_OBProgram()` to :
 - Set/Reset the write protection per bank
 - Set the Read protection Level
 - Set the BOR level
 - Program the user Option Bytes
 - PCROP protection configuration and control per bank
 - Secure area configuration and control per bank
 - Core Boot address configuration
3. FLASH Memory Lock and unlock per Bank: `HAL_FLASHEx_Lock_Bank1` and `HAL_FLASHEx_Unlock_Bank1` functions

25.2.3 Detailed description of functions

`HAL_FLASHEx_Erase`

Function name

`HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraselTypeDef * pEraselInit, uint32_t * SectorError)`

Function description

Perform a mass erase or erase the specified FLASH memory sectors.

Parameters

- **pEraselInit:** pointer to an `FLASH_EraselTypeDef` structure that contains the configuration information for the erasing.
- **SectorError:** pointer to variable that contains the configuration information on faulty sector in case of error (0xFFFFFFFF means that all the sectors have been correctly erased)

Return values

- **HAL:** Status

HAL_FLASHEx_Erase_IT

Function name

`HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)`

Function description

Perform a mass erase or erase the specified FLASH memory sectors with interrupt enabled.

Parameters

- **pEraseInit:** pointer to an `FLASH_EraseInitTypeDef` structure that contains the configuration information for the erasing.

Return values

- **HAL:** Status

HAL_FLASHEx_OBProgram

Function name

`HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)`

Function description

Program option bytes.

Parameters

- **pOBInit:** pointer to an `FLASH_OBInitStruct` structure that contains the configuration information for the programming.

Return values

- **HAL:** Status

HAL_FLASHEx_OBGetConfig

Function name

`void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)`

Function description

Get the Option byte configuration.

Parameters

- **pOBInit:** pointer to an `FLASH_OBInitStruct` structure that contains the configuration information for the programming.

Return values

- **None:**

Notes

- The parameter Banks of the `pOBInit` structure must be exclusively `FLASH_BANK_1` or `FLASH_BANK_2` as this parameter is used to get the given Bank WRP, PCROP and secured area.

HAL_FLASHEx_Unlock_Bank1

Function name

`HAL_StatusTypeDef HAL_FLASHEx_Unlock_Bank1 (void)`

Function description

Unlock the FLASH Bank1 control registers access.

Return values

- **HAL:** Status

HAL_FLASHEx_Lock_Bank1**Function name****HAL_StatusTypeDef HAL_FLASHEx_Lock_Bank1 (void)****Function description**

Locks the FLASH Bank1 control registers access.

Return values

- **HAL:** Status

HAL_FLASHEx_Unlock_Bank2**Function name****HAL_StatusTypeDef HAL_FLASHEx_Unlock_Bank2 (void)****Function description**

Unlock the FLASH Bank2 control registers access.

Return values

- **HAL:** Status

HAL_FLASHEx_Lock_Bank2**Function name****HAL_StatusTypeDef HAL_FLASHEx_Lock_Bank2 (void)****Function description**

Locks the FLASH Bank2 control registers access.

Return values

- **HAL:** Status

FLASH_Erase_Sector**Function name****void FLASH_Erase_Sector (uint32_t Sector, uint32_t Bank, uint32_t VoltageRange)****Function description**

Erase the specified FLASH memory sector.

Parameters

- **Sector:** FLASH sector to erase
- **Banks:** Banks to be erased This parameter can be one of the following values:
 - FLASH_BANK_1: Bank1 to be erased
 - FLASH_BANK_2: Bank2 to be erased
 - FLASH_BANK_BOTH: Bank1 and Bank2 to be erased
- **VoltageRange:** The device program/erase parallelism. This parameter can be one of the following values:
 - FLASH_VOLTAGE_RANGE_1 : Flash program/erase by 8 bits
 - FLASH_VOLTAGE_RANGE_2 : Flash program/erase by 16 bits
 - FLASH_VOLTAGE_RANGE_3 : Flash program/erase by 32 bits
 - FLASH_VOLTAGE_RANGE_4 : Flash program/erase by 62 bits

Return values

- None:

25.3 FLASHEx Firmware driver defines

25.3.1 FLASHEx

FLASH Banks

FLASH_BANK_1

Bank 1

FLASH_BANK_2

Bank 2

FLASH_BANK_BOTH

Bank1 and Bank2

FLASH Boot Address

OB_BOOTADDR_ITCM_RAM

Boot from ITCM RAM (0x00000000)

OB_BOOTADDR_SYSTEM

Boot from System memory bootloader (0x00100000)

OB_BOOTADDR_ITCM_FLASH

Boot from Flash on ITCM interface (0x00200000)

OB_BOOTADDR_AXIM_FLASH

Boot from Flash on AXIM interface (0x08000000)

OB_BOOTADDR_DTCM_RAM

Boot from DTCM RAM (0x20000000)

OB_BOOTADDR_SRAM1

Boot from SRAM1 (0x20010000)

OB_BOOTADDR_SRAM2

Boot from SRAM2 (0x2004C000)

FLASH BOR Reset Level

OB_BOR_LEVEL3

Supply voltage ranges from 2.70 to 3.60 V

OB_BOR_LEVEL2

Supply voltage ranges from 2.40 to 2.70 V

OB_BOR_LEVEL1

Supply voltage ranges from 2.10 to 2.40 V

OB_BOR_OFF

Supply voltage ranges from 1.62 to 2.10 V

FLASHEx Private macros to check input parameters

IS_FLASH_TYPEERASE

IS_VOLTAGERANGE

IS_WRPSTATE

IS_OPTIONBYTE

IS_OB_BOOT_ADDRESS

IS_OB_RDP_LEVEL

IS_OB_WWDG_SOURCE

IS_OB_IWDG_SOURCE

IS_OB_STOP_SOURCE

IS_OB_STDBY_SOURCE

IS_OB_IWDG_STOP_FREEZE

IS_OB_IWDG_STDBY_FREEZE

IS_OB_BOR_LEVEL

IS_FLASH_LATENCY

IS_FLASH_ADDRESS

IS_FLASH_NBSECTORS

IS_FLASH_SECTOR

IS_OB_WRP_SECTOR

IS_FLASH_BANK

IS_OB_PCROP_RDP

IS_OB_SECURE_RDP

IS_OB_USER_SWAP_BANK

IS_OB_USER_IOHSLV

IS_OB_IWDG1_SOURCE

IS_OB_STOP_D1_RESET

IS_OB_STDBY_D1_RESET

IS_OB_USER_IWDG_STOP

IS_OB_USER_IWDG_STDBY

IS_OB_USER_SECURITY

IS_OB_USER_TYPE

IS_OB_BOOT_ADD_OPTION

FLASHEx OB BOOT OPTION

OB_BOOT_ADD0

Select Boot Address 0

OB_BOOT_ADD1

Select Boot Address 1

OB_BOOT_ADD_BOTH

Select Boot Address 0 and 1

FLASHEx OB IOHSLV

OB_IOHSLV_DISABLE

IOHSLV disabled

OB_IOHSLV_ENABLE

IOHSLV enabled

FLASHEx OB IWDG1 SW

OB_IWDG1_SW

Hardware independent watchdog 1

OB_IWDG1_HW

Software independent watchdog 1

FLASHEx OB NRST STDBY D1

OB_STDBY_RST_D1

Reset generated when entering the D1 to standby mode

OB_STDBY_NO_RST_D1

No reset generated when entering the D1 to standby mode

FLASHEx OB NRST STOP D1

OB_STOP_RST_D1

Reset generated when entering the D1 to stop mode

OB_STOP_NO_RST_D1

No reset generated when entering the D1 to stop mode

FLASHEx OB PCROP RDP

OB_PCROP_RDP_NOT_ERASE

PCROP area is not erased when the RDP level is decreased from Level 1 to Level 0 or during a mass erase

OB_PCROP_RDP_ERASE

PCROP area is erased when the RDP level is decreased from Level 1 to Level 0 (full mass erase)

FLASHEx OB SECURE RDP

OB_SECURE_RDP_NOT_ERASE

Secure area is not erased when the RDP level is decreased from Level 1 to Level 0 or during a mass erase

OB_SECURE_RDP_ERASE

Secure area is erased when the RDP level is decreased from Level 1 to Level 0 (full mass erase)

FLASHEx OB SECURITY**OB_SECURITY_DISABLE**

security enabled

OB_SECURITY_ENABLE

security disabled

FLASHEx OB SWAP BANK**OB_SWAP_BANK_DISABLE**

Bank swap disabled

OB_SWAP_BANK_ENABLE

Bank swap enabled

FLASHEx OB USER Type**OB_USER_NRST_STOP_D1**

Reset when entering Stop mode selection

OB_USER_NRST_STDBY_D1

Reset when entering standby mode selection

OB_USER_IWDG_STOP

Independent watchdog counter freeze in stop mode

OB_USER_IWDG_STDBY

Independent watchdog counter freeze in standby mode

OB_USER_ST_RAM_SIZE

dedicated DTCM Ram size selection

OB_USER_SECURITY

security selection

OB_USER_SWAP_BANK

Bank swap selection

OB_USER_IOHSLV

IO HSLV selection

OB_USER_IWDG1_SW

Independent watchdog selection

FLASH Option Bytes IWatchdog**OB_IWDG_SW**

Software IWDG selected

OB_IWDG_HW

Hardware IWDG selected

FLASH IWDG Counter Freeze in STANDBY**OB_IWDG_STDBY_FREEZE**

Freeze IWDG counter in STANDBY mode

OB_IWDG_STDBY_ACTIVE

IWDG counter active in STANDBY mode

FLASH IWDG Counter Freeze in STOP**OB_IWDG_STOP_FREEZE**

Freeze IWDG counter in STOP mode

OB_IWDG_STOP_ACTIVE

IWDG counter active in STOP mode

FLASH Option Bytes nRST_STDBY**OB_STDBY_NO_RST**

No reset generated when entering in STANDBY

OB_STDBY_RST

Reset generated when entering in STANDBY

FLASH Option Bytes nRST_STOP**OB_STOP_NO_RST**

No reset generated when entering in STOP

OB_STOP_RST

Reset generated when entering in STOP

FLASH Option Bytes Read Protection**OB_RDP_LEVEL_0****OB_RDP_LEVEL_1****OB_RDP_LEVEL_2**

Warning: When enabling read protection level 2 it's no more possible to go back to level 1 or 0

FLASH Option Bytes Write Protection**OB_WRP_SECTOR_0**

Write protection of Sector0

OB_WRP_SECTOR_1

Write protection of Sector1

OB_WRP_SECTOR_2

Write protection of Sector2

OB_WRP_SECTOR_3

Write protection of Sector3

OB_WRP_SECTOR_4

Write protection of Sector4

OB_WRP_SECTOR_5

Write protection of Sector5

OB_WRP_SECTOR_6

Write protection of Sector6

OB_WRP_SECTOR_7

Write protection of Sector7

OB_WRP_SECTOR_All

Write protection of all Sectors

FLASH Option Bytes WWatchdog**OB_WWDG_SW**

Software WWDG selected

OB_WWDG_HW

Hardware WWDG selected

FLASH Option Type**OPTIONBYTE_WRP**

WRP option byte configuration

OPTIONBYTE_RDP

RDP option byte configuration

OPTIONBYTE_USER

USER option byte configuration

OPTIONBYTE_PCROP

PCROP option byte configuration

OPTIONBYTE_BOR

BOR option byte configuration

OPTIONBYTE_SECURE_AREA

secure area option byte configuration

OPTIONBYTE_BOOTADD

BOOT ADD option byte configuration

FLASH Type Erase**FLASH_TYPEERASE_SECTORS**

Sectors erase only

FLASH_TYPEERASE_MASSERASE

Flash Mass erase activation

FLASH Voltage Range**FLASH_VOLTAGE_RANGE_1**

Flash program/erase by 8 bits

FLASH_VOLTAGE_RANGE_2

Flash program/erase by 16 bits

FLASH_VOLTAGE_RANGE_3

Flash program/erase by 32 bits

FLASH_VOLTAGE_RANGE_4

Flash program/erase by 64 bits

FLASH WRP State**OB_WRPSTATE_DISABLE**

Disable the write protection of the desired bank 1 sectors

OB_WRPSTATE_ENABLE

Enable the write protection of the desired bank 1 sectors

26 HAL GPIO Generic Driver

26.1 GPIO Firmware driver registers structures

26.1.1 GPIO_InitTypeDef

Data Fields

- *uint32_t Pin*
- *uint32_t Mode*
- *uint32_t Pull*
- *uint32_t Speed*
- *uint32_t Alternate*

Field Documentation

- *uint32_t GPIO_InitTypeDef::Pin*
Specifies the GPIO pins to be configured. This parameter can be any value of **GPIO pins define**
- *uint32_t GPIO_InitTypeDef::Mode*
Specifies the operating mode for the selected pins. This parameter can be a value of **GPIO mode define**
- *uint32_t GPIO_InitTypeDef::Pull*
Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of **GPIO pull define**
- *uint32_t GPIO_InitTypeDef::Speed*
Specifies the speed for the selected pins. This parameter can be a value of **GPIO speed define**
- *uint32_t GPIO_InitTypeDef::Alternate*
Peripheral to be connected to the selected pins. This parameter can be a value of **GPIO Alternate Function Selection**

26.2 GPIO Firmware driver API description

26.2.1 GPIO Peripheral features

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes:

- Input mode
- Analog mode
- Output mode
- Alternate function mode
- External interrupt/event lines

During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.

All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.

In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.

All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

The external interrupt/event controller consists of up to 23 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

26.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: `__HAL_RCC_GPIOx_CLK_ENABLE()`.
2. Configure the GPIO pin(s) using `HAL_GPIO_Init()`.
 - Configure the IO mode using "Mode" member from `GPIO_InitTypeDef` structure
 - Activate Pull-up, Pull-down resistor using "Pull" member from `GPIO_InitTypeDef` structure.
 - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from `GPIO_InitTypeDef` structure.
 - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from `GPIO_InitTypeDef` structure.
 - Analog mode is required when a pin is to be used as ADC channel or DAC output.
 - In case of external interrupt/event selection the "Mode" member from `GPIO_InitTypeDef` structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using `HAL_NVIC_SetPriority()` and enable it using `HAL_NVIC_EnableIRQ()`.
4. To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()`.
5. To set/reset the level of a pin configured in output mode use `HAL_GPIO_WritePin()`/`HAL_GPIO_TogglePin()`.
6. To lock pin configuration until next reset use `HAL_GPIO_LockPin()`.
7. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
8. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
9. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

26.2.3 Initialization and de-initialization functions

This section provides functions allowing to initialize and de-initialize the GPIOs to be ready for use.

This section contains the following APIs:

- [`HAL_GPIO_Init`](#)
- [`HAL_GPIO_DelInit`](#)

26.2.4 IO operation functions

This section contains the following APIs:

- [`HAL_GPIO_ReadPin`](#)
- [`HAL_GPIO_WritePin`](#)
- [`HAL_GPIO_TogglePin`](#)
- [`HAL_GPIO_LockPin`](#)
- [`HAL_GPIO_EXTI_IRQHandler`](#)
- [`HAL_GPIO_EXTI_Callback`](#)

26.2.5 Detailed description of functions

HAL_GPIO_Init

Function name

`void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)`

Function description

Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral.
- **GPIO_InitStruct:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **None:**

HAL_GPIO_DeInit

Function name

`void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)`

Function description

De-initializes the GPIOx peripheral registers to their default reset values.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_ReadPin

Function name

`GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)`

Function description

Reads the specified input port pin.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).

Return values

- **The:** input port pin value.

HAL_GPIO_WritePin

Function name

`void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)`

Function description

Sets or clears the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
 - GPIO_PIN_RESET: to clear the port pin
 - GPIO_PIN_SET: to set the port pin

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

HAL_GPIO_TogglePin

Function name

```
void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
```

Function description

Toggles the specified GPIO pins.

Parameters

- **GPIOx:** Where x can be (A..K) to select the GPIO peripheral.
- **GPIO_Pin:** Specifies the pins to be toggled.

Return values

- **None:**

HAL_GPIO_LockPin

Function name

```
HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
```

Function description

Locks GPIO Pins configuration registers.

Parameters

- **GPIOx:** where x can be (A..K) to select the GPIO peripheral for STM32H7 family
- **GPIO_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

Notes

- The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name

```
void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)
```

Function description

Handle EXTI interrupt request.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Callback

Function name

void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)

Function description

EXTI line detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

26.3 GPIO Firmware driver defines

26.3.1 GPIO

GPIO Alternate Function Selection

GPIO_AF0_RTC_50Hz

GPIO_AF0_MCO

GPIO_AF0_SWJ

GPIO_AF0_LCDBIAS

GPIO_AF0_TRACE

GPIO_AF1_TIM1

GPIO_AF1_TIM2

GPIO_AF1_TIM16

GPIO_AF1_TIM17

GPIO_AF1_LPTIM1

GPIO_AF1_HRTIM1

GPIO_AF2_TIM3

GPIO_AF2_TIM4

GPIO_AF2_TIM5
GPIO_AF2_TIM12
GPIO_AF2_HRTIM1
GPIO_AF2_SAI1
GPIO_AF3_TIM8
GPIO_AF3_LPTIM2
GPIO_AF3_DFSDM1
GPIO_AF3_HRTIM1
GPIO_AF3_LPTIM3
GPIO_AF3_LPTIM4
GPIO_AF3_LPTIM5
GPIO_AF3_LPUART
GPIO_AF4_I2C1
GPIO_AF4_I2C2
GPIO_AF4_I2C3
GPIO_AF4_I2C4
GPIO_AF4_TIM15
GPIO_AF4_CEC
GPIO_AF4_LPTIM2
GPIO_AF4_USART1
GPIO_AF4_DFSDM1
GPIO_AF5_SPI1
GPIO_AF5_SPI2
GPIO_AF5_SPI3
GPIO_AF5_SPI4
GPIO_AF5_SPI5
GPIO_AF5_SPI6
GPIO_AF5_CEC

GPIO_AF6_SPI2
GPIO_AF6_SPI3
GPIO_AF6_SAI1
GPIO_AF6_SAI3
GPIO_AF6_I2C4
GPIO_AF6_DFSDM1
GPIO_AF6_UART4
GPIO_AF7_SPI2
GPIO_AF7_SPI3
GPIO_AF7_SPI6
GPIO_AF7_USART1
GPIO_AF7_USART2
GPIO_AF7_USART3
GPIO_AF7_USART6
GPIO_AF7_UART7
GPIO_AF7_DFSDM1
GPIO_AF7_SDIO1
GPIO_AF8_SPI6
GPIO_AF8_SAI2
GPIO_AF8_SAI4
GPIO_AF8_USART4
GPIO_AF8_USART5
GPIO_AF8_USART8
GPIO_AF8_SPDIF
GPIO_AF8_LPUART
GPIO_AF8_SDIO1
GPIO_AF9_FDCAN1
GPIO_AF9_FDCAN2

GPIO_AF9_TIM13
GPIO_AF9_TIM14
GPIO_AF9_QUADSPI
GPIO_AF9_SDIO2
GPIO_AF9_LTDC
GPIO_AF9_SPDIF
GPIO_AF9_FMC
GPIO_AF9_SAI4
GPIO_AF10_QUADSPI
GPIO_AF10_SAI2
GPIO_AF10_SAI4
GPIO_AF10_SDIO2
GPIO_AF10_OTG2_HS
GPIO_AF10_OTG1_FS
GPIO_AF10_COMP1
GPIO_AF10_COMP2
GPIO_AF10_LTDC
GPIO_AF11_SWP
GPIO_AF11_ETH
GPIO_AF11_MDIOS
GPIO_AF11_OTG1_HS
GPIO_AF11_UART7
GPIO_AF11_SDIO2
GPIO_AF11_DFSDM1
GPIO_AF11_COMP1
GPIO_AF11_COMP2
GPIO_AF11_I2C4
GPIO_AF12_FMC

[GPIO_AF12_SDIO1](#)

[GPIO_AF12_MDIOS](#)

[GPIO_AF12_OTG2_FS](#)

[GPIO_AF12_COMP1](#)

[GPIO_AF12_COMP2](#)

[GPIO_AF12_LTDC](#)

[GPIO_AF13_DCMI](#)

[GPIO_AF13_COMP1](#)

[GPIO_AF13_COMP2](#)

[GPIO_AF13_LTDC](#)

[GPIO_AF14_LTDC](#)

[GPIO_AF14_UART5](#)

[GPIO_AF15_EVENTOUT](#)

[IS_GPIO_AF](#)

GPIO Exported Macros

[__HAL_GPIO_EXTI_GET_FLAG](#)

Description:

- Checks whether the specified EXTI line flag is set or not.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI line flag to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

- The: new state of `__EXTI_LINE__` (SET or RESET).

[__HAL_GPIO_EXTI_CLEAR_FLAG](#)

Description:

- Clears the EXTI's line pending flags.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI lines flags to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

Return value:

- None

[__HAL_GPIO_EXTI_GET_IT](#)

Description:

- Checks whether the specified EXTI line is asserted or not.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

- The: new state of `__EXTI_LINE__` (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_IT**Description:**

- Clears the EXTI's line pending bits.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI lines to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GENERATE_SWIT**Description:**

- Generates a Software interrupt on selected EXTI line.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

- None

GPIO mode define**GPIO_MODE_INPUT**

Input Floating Mode

GPIO_MODE_OUTPUT_PP

Output Push Pull Mode

GPIO_MODE_OUTPUT_OD

Output Open Drain Mode

GPIO_MODE_AF_PP

Alternate Function Push Pull Mode

GPIO_MODE_AF_OD

Alternate Function Open Drain Mode

GPIO_MODE_ANALOG

Analog Mode

GPIO_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

GPIO_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

GPIO_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

GPIO_MODE_EVT_RISING

External Event Mode with Rising edge trigger detection

GPIO_MODE_EVT_FALLING

External Event Mode with Falling edge trigger detection

GPIO_MODE_EVT_RISING_FALLING

External Event Mode with Rising/Falling edge trigger detection

GPIO pins define**GPIO_PIN_0****GPIO_PIN_1****GPIO_PIN_2****GPIO_PIN_3****GPIO_PIN_4****GPIO_PIN_5****GPIO_PIN_6****GPIO_PIN_7****GPIO_PIN_8****GPIO_PIN_9****GPIO_PIN_10****GPIO_PIN_11****GPIO_PIN_12****GPIO_PIN_13****GPIO_PIN_14****GPIO_PIN_15****GPIO_PIN_All****GPIO_PIN_MASK*****GPIO pull define*****GPIO_NOPULL**

No Pull-up or Pull-down activation

GPIO_PULLUP

Pull-up activation

GPIO_PULLDOWN

Pull-down activation

GPIO speed define**GPIO_SPEED_FREQ_LOW**

Low speed

GPIO_SPEED_FREQ_MEDIUM

Medium speed

GPIO_SPEED_FREQ_HIGH

Fast speed

GPIO_SPEED_FREQ VERY HIGH

High speed

27 HAL GPIO Extension Driver

27.1 GPIOEx Firmware driver defines

27.1.1 GPIOEx

GPIO Get Port Index

`GPIO_GET_INDEX`

28 HAL HASH Generic Driver

28.1 HASH Firmware driver registers structures

28.1.1 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 input 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.

28.1.2 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`

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

28.2 HASH Firmware driver API description

28.2.1 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_MsInit()` 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.

28.2.2

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_DelInit**](#)
- [**HAL_HASH_MspInit**](#)
- [**HAL_HASH_MspDelInit**](#)
- [**HAL_HASH_InCpltCallback**](#)
- [**HAL_HASH_DgstCpltCallback**](#)
- [**HAL_HASH_ErrorCallback**](#)

28.2.3 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**](#)

28.2.4 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*](#)

28.2.5 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*](#)

28.2.6 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*](#)

28.2.7 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*](#)

28.2.8 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](#)

28.2.9 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](#)

28.2.10 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_DelInit

Function name

```
HAL_StatusTypeDef HAL_HASH_DelInit (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_MspDelInit

Function name

```
void HAL_HASH_MspDelInit (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 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.

28.3 HASH Firmware driver defines

28.3.1 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 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

29 HAL HASH Extension Driver

29.1 HASHEx Firmware driver API description

29.1.1 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.

29.1.2 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*](#)

29.1.3 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*](#)

29.1.4 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*](#)

29.1.5 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_HMACEEx_SHA256_Start*](#)

29.1.6 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_HMACEEx_SHA224_Start_IT\(\)*](#)
- SHA256
 - [*HAL_HMACEEx_SHA256_Start_IT\(\)*](#)

This section contains the following APIs:

- [*HAL_HMACEEx_SHA224_Start_IT*](#)
- [*HAL_HMACEEx_SHA256_Start_IT*](#)

29.1.7 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_HMACEEx_SHA224_Start_DMA\(\)*](#)
- SHA256
 - [*HAL_HMACEEx_SHA256_Start_DMA\(\)*](#)

When resorting to DMA mode to enter the data in the IP for HMAC processing, user must resort to [*HAL_HMACEEx_xxx_Start_DMA\(\)*](#) then read the resulting digest with [*HAL_HASHEx_xxx_Finish\(\)*](#).

This section contains the following APIs:

- [*HAL_HMACEEx_SHA224_Start_DMA*](#)
- [*HAL_HMACEEx_SHA256_Start_DMA*](#)

29.1.8 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_HMACEEx_MD5_Step1_2_DMA\(\)*](#)
 - [*HAL_HMACEEx_MD5_Step2_DMA\(\)*](#)
 - [*HAL_HMACEEx_MD5_Step2_3_DMA\(\)*](#)
- SHA1
 - [*HAL_HMACEEx_SHA1_Step1_2_DMA\(\)*](#)
 - [*HAL_HMACEEx_SHA1_Step2_DMA\(\)*](#)
 - [*HAL_HMACEEx_SHA1_Step2_3_DMA\(\)*](#)
- SHA256
 - [*HAL_HMACEEx_SHA224_Step1_2_DMA\(\)*](#)
 - [*HAL_HMACEEx_SHA224_Step2_DMA\(\)*](#)
 - [*HAL_HMACEEx_SHA224_Step2_3_DMA\(\)*](#)
- SHA256
 - [*HAL_HMACEEx_SHA256_Step1_2_DMA\(\)*](#)
 - [*HAL_HMACEEx_SHA256_Step2_DMA\(\)*](#)
 - [*HAL_HMACEEx_SHA256_Step2_3_DMA\(\)*](#)

User must first start-up the multi-buffer DMA-based HMAC computation in calling [*HAL_HMACEEx_xxx_Step1_2_DMA\(\)*](#). This carries out HMAC step 1 and initiates 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](#)

29.1.9 Detailed description of functions

HAL_HASHEx_SHA224_Start

Function name

```
HAL_StatusTypeDef HAL_HASHEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer,  
uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
```

Function description

Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Accumulate

Function name

```
HAL_StatusTypeDef HAL_HASHEx_SHA224_Accumulate (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

If not already done, initialize the HASH peripheral in SHA224 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA224_Accumulate() can be used to feed several input buffers back-to-back to the IP that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA224_Start().
- Field hhash->Phase of HASH handle is tested to check whether or not the IP has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASHEx_SHA224_Start() to read it, feeding at the same time the last input buffer to the IP.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA224_Start() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA256_Start

Function name

```
HAL_StatusTypeDef HAL_HASHEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer,  
uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
```

Function description

Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Accumulate

Function name

```
HAL_StatusTypeDef HAL_HASHEx_SHA256_Accumulate (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

If not already done, initialize the HASH peripheral in SHA256 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).

- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA256_Accumulate() can be used to feed several input buffers back-to-back to the IP that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA256_Start().
- Field hhash->Phase of HASH handle is tested to check whether or not the IP has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASHEx_SHA256_Start() to read it, feeding at the same time the last input buffer to the IP.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA256_Start() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA224_Start_IT

Function name

```
HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer,  
uint32_t Size, uint8_t * pOutBuffer)
```

Function description

Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Start_IT

Function name

```
HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer,  
uint32_t Size, uint8_t * pOutBuffer)
```

Function description

Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in SHA224 mode then initiate a DMA transfer to feed the input buffer to the IP.

Parameters

- hhash:** HASH handle.
- pInBuffer:** pointer to the input buffer (buffer to be hashed).
- Size:** length of the input buffer in bytes.

Return values

- HAL:** status

Notes

- Once the DMA transfer is finished, **HAL_HASHEx_SHA224_Finish()** API must be called to retrieve the computed digest.

HAL_HASHEx_SHA224_Finish

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in SHA224 mode.

Parameters

- hhash:** HASH handle.
- pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- Timeout:** Timeout value.

Return values

- HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASHEx_SHA224_Finish()** can be used as well to retrieve the digest in HMAC SHA224 mode.

HAL_HASHEx_SHA256_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in SHA256 mode then initiate a DMA transfer to feed the input buffer to the IP.

Parameters

- hhash:** HASH handle.

- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HASHEx_SHA256_Finish

Function name

```
HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer,  
uint32_t Timeout)
```

Function description

Return the computed digest in SHA256 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASHEx_SHA256_Finish() can be used as well to retrieve the digest in HMAC SHA256 mode.

HAL_HMACEx_SHA224_Start

Function name

```
HAL_StatusTypeDef HAL_HMACEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer,  
uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
```

Function description

Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEEx_SHA256_Start

Function name

```
HAL_StatusTypeDef HAL_HMACEEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer,  
uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
```

Function description

Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEEx_SHA224_Start_IT

Function name

```
HAL_StatusTypeDef HAL_HMACEEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer,  
uint32_t Size, uint8_t * pOutBuffer)
```

Function description

Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEEx_SHA256_Start_IT

Function name

```
HAL_StatusTypeDef HAL_HMACEEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer,  
uint32_t Size, uint8_t * pOutBuffer)
```

Function description

Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA224_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the IP.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA224_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMACEx_SHA256_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the IP.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMACEx_MD5_Step1_2_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEx_MD5_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

MD5 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the IP, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the IP. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_MD5_Step2_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEx_MD5_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

MD5 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).

- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the IP.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_MD5_Step2_3_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEx_MD5_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

MD5 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the IP, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA1_Step1_2_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEx_SHA1_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

SHA1 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the IP, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the IP. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

`HAL_HMACEx_SHA1_Step2_DMA`

Function name

```
HAL_StatusTypeDef HAL_HMACEx_SHA1_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
    pInBuffer, uint32_t Size)
```

Function description

SHA1 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the IP.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in `hhash->Init.pKey` and `hhash->Init.KeySize`.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

`HAL_HMACEx_SHA1_Step2_3_DMA`

Function name

```
HAL_StatusTypeDef HAL_HMACEx_SHA1_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
    pInBuffer, uint32_t Size)
```

Function description

SHA1 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the IP, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.

- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA224_Step1_2_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEx_SHA224_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
```

Function description

SHA224 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the IP, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the IP. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA224_Step2_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEx_SHA224_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
```

Function description

SHA224 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the IP.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEEx_SHA224_Step2_3_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEEx_SHA224_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

SHA224 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the IP, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEEx_SHA256_Step1_2_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEEx_SHA256_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

SHA256 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the IP, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the IP. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEEx_SHA256_Step2_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEEx_SHA256_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

SHA256 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the IP.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEEx_SHA256_Step2_3_DMA

Function name

```
HAL_StatusTypeDef HAL_HMACEEx_SHA256_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t *  
pInBuffer, uint32_t Size)
```

Function description

SHA256 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the IP, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

30 HAL HCD Generic Driver

30.1 HCD Firmware driver registers structures

30.1.1 HCD_HandleTypeDef

Data Fields

- *HCD_TypeDef * Instance*
- *HCD_InitTypeDef Init*
- *HCD_HCTypedef hc*
- *HAL_LockTypeDef Lock*
- *_IO HCD_StateTypeDef State*
- *void * pData*

Field Documentation

- ***HCD_TypeDef* HCD_HandleTypeDef::Instance***
Register base address
- ***HCD_InitTypeDef HCD_HandleTypeDef::Init***
HCD required parameters
- ***HCD_HCTypedef HCD_HandleTypeDef::hc[15]***
Host channels parameters
- ***HAL_LockTypeDef HCD_HandleTypeDef::Lock***
HCD peripheral status
- ***_IO HCD_StateTypeDef HCD_HandleTypeDef::State***
HCD communication state
- ***void* HCD_HandleTypeDef::pData***
Pointer Stack Handler

30.2 HCD Firmware driver API description

30.2.1 How to use this driver

1. Declare a HCD_HandleTypeDef handle structure, for example: `HCD_HandleTypeDef hhcd;`
2. Fill parameters of Init structure in HCD handle
3. Call `HAL_HCD_Init()` API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the `HAL_HCD_MspInit()` API:
 - a. Enable the HCD/USB Low Level interface clock using the following macros
 - `__OTGFS-OTG_CLK_ENABLE()` or `__OTGHS-OTG_CLK_ENABLE()`
 - `__OTGHSULPI_CLK_ENABLE()` For High Speed Mode
 - b. Initialize the related GPIO clocks
 - c. Configure HCD pin-out
 - d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
 - a. `hhcd.pData = phost;`

6. Enable HCD transmission and reception:
 - a. `HAL_HCD_Start();`

30.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- [`HAL_HCD_Init`](#)
- [`HAL_HCD_HC_Init`](#)
- [`HAL_HCD_HC_Halt`](#)
- [`HAL_HCD_DelInit`](#)
- [`HAL_HCD_MspInit`](#)
- [`HAL_HCD_MspDelInit`](#)

30.2.3 IO operation functions

This section contains the following APIs:

- [`HAL_HCD_HC_SubmitRequest`](#)
- [`HAL_HCD_IRQHandler`](#)
- [`HAL_HCD_SOF_Callback`](#)
- [`HAL_HCD_Connect_Callback`](#)
- [`HAL_HCD_Disconnect_Callback`](#)
- [`HAL_HCD_HC_NotifyURBChange_Callback`](#)

30.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.

This section contains the following APIs:

- [`HAL_HCD_Start`](#)
- [`HAL_HCD_Stop`](#)
- [`HAL_HCD_ResetPort`](#)

30.2.5 Peripheral State functions

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

This section contains the following APIs:

- [`HAL_HCD_GetState`](#)
- [`HAL_HCD_HC_GetURBState`](#)
- [`HAL_HCD_HC_GetXferCount`](#)
- [`HAL_HCD_HC_GetState`](#)
- [`HAL_HCD_GetCurrentFrame`](#)
- [`HAL_HCD_GetCurrentSpeed`](#)

30.2.6 Detailed description of functions

`HAL_HCD_Init`

Function name

`HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)`

Function description

Initialize the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_DelInit

Function name

HAL_StatusTypeDef HAL_HCD_DelInit (HCD_HandleTypeDef * hhcd)

Function description

DeInitialize the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_HC_Init

Function name

HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epxnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)

Function description

Initialize a host channel.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **epxnum:** Endpoint number. This parameter can be a value from 1 to 15
- **dev_address:** Current device address This parameter can be a value from 0 to 255
- **speed:** Current device speed. This parameter can be one of these values: HCD_SPEED_HIGH: High speed mode, HCD_SPEED_FULL: Full speed mode, HCD_SPEED_LOW: Low speed mode
- **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type, EP_TYPE_ISOC: Isochronous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type
- **mps:** Max Packet Size. This parameter can be a value from 0 to32K

Return values

- **HAL:** status

HAL_HCD_HC_Halt

Function name

HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

Halt a host channel.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15

Return values

- **HAL:** status

HAL_HCD_MspInit

Function name

```
void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)
```

Function description

Initializes the HCD MSP.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_MspDeInit

Function name

```
void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)
```

Function description

DeInitializes HCD MSP.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_HC_SubmitRequest

Function name

```
HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num,  
uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)
```

Function description

Submit a new URB for processing.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **direction:** Channel number. This parameter can be one of these values: 0 : Output / 1 : Input
- **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type/ EP_TYPE_ISOC: Isochronous type/ EP_TYPE_BULK: Bulk type/ EP_TYPE_INTR: Interrupt type/
- **token:** Endpoint Type. This parameter can be one of these values: 0: HC_PID_SETUP / 1: HC_PID_DATA1
- **pbuff:** pointer to URB data
- **length:** Length of URB data
- **do_ping:** activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active

Return values

- **HAL:** status

HAL_HCD_IRQHandler

Function name

```
void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)
```

Function description

This function handles HCD interrupt request.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_SOF_Callback

Function name

```
void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)
```

Function description

SOF callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_Connect_Callback

Function name

```
void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)
```

Function description

Connexion Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_Disconnect_Callback

Function name

```
void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)
```

Function description

Disconnection Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_HC_NotifyURBChange_Callback

Function name

```
void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum,  
HCD_URBStateTypeDef urb_state)
```

Function description

Notify URB state change callback.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15
- **urb_state:** This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/

Return values

- **None:**

HAL_HCD_ResetPort

Function name

```
HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * hhcd)
```

Function description

Reset the host port.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Start

Function name

```
HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * hhcd)
```

Function description

Start the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Stop

Function name

```
HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * hhcd)
```

Function description

Stop the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_GetState**Function name**

HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef * hhcd)

Function description

Return the HCD state.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** state

HAL_HCD_HC_GetURBState**Function name**

HCD_URBStateTypeDef HAL_HCD_HC_GetURBState (HCD_HandleTypeDef * hhcd, uint8_t chnum)

Function description

Return URB state for a channel.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **URB:** state. This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/

HAL_HCD_HC_GetXferCount**Function name**

uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef * hhcd, uint8_t chnum)

Function description

Return the last host transfer size.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **last:** transfer size in byte

HAL_HCD_HC_GetState**Function name**

HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef * hhcd, uint8_t chnum)

Function description

Return the Host Channel state.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **Host:** channel state This parameter can be one of the these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR/

HAL_HCD_GetCurrentFrame**Function name**

```
uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)
```

Function description

Return the current Host frame number.

Parameters

- **hhcd:** HCD handle

Return values

- **Current:** Host frame number

HAL_HCD_GetCurrentSpeed**Function name**

```
uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)
```

Function description

Return the Host enumeration speed.

Parameters

- **hhcd:** HCD handle

Return values

- **Enumeration:** speed

30.3 HCD Firmware driver defines

30.3.1 HCD

HCD Exported Macros

`_HAL_HCD_ENABLE`

`_HAL_HCD_DISABLE`

`_HAL_HCD_GET_FLAG`

`_HAL_HCD_CLEAR_FLAG`

`_HAL_HCD_IS_INVALID_INTERRUPT`

`_HAL_HCD_CLEAR_HC_INT`

`_HAL_HCD_MASK_HALT_HC_INT`

`_HAL_HCD_UNMASK_HALT_HC_INT`

`_HAL_HCD_MASK_ACK_HC_INT`

`_HAL_HCD_UNMASK_ACK_HC_INT`

HCD Instance definition

`IS_HCD_ALL_INSTANCE`

HCD PHY Module

`HCD_PHY_ULPI`

`HCD_PHY_EMBEDDED`

HCD Speed

`HCD_SPEED_HIGH`

`HCD_SPEED_LOW`

`HCD_SPEED_FULL`

31 HAL HRTIM Generic Driver

31.1 HRTIM Firmware driver registers structures

31.1.1 HRTIM_InitTypeDef

Data Fields

- `uint32_t HRTIMInterruptRequests`
- `uint32_t SyncOptions`
- `uint32_t SyncInputSource`
- `uint32_t SyncOutputSource`
- `uint32_t SyncOutputPolarity`

Field Documentation

- `uint32_t HRTIM_InitTypeDef::HRTIMInterruptRequests`

Specifies which interrupts requests must enabled for the HRTIM instance This parameter can be any combination of **HRTIM Common Interrupt Enable**

- `uint32_t HRTIM_InitTypeDef::SyncOptions`

Specifies how the HRTIM instance handles the external synchronization signals This parameter can be a combination of **HRTIM Synchronization Options**

- `uint32_t HRTIM_InitTypeDef::SyncInputSource`

Specifies the external synchronization input source This parameter can be a value of **HRTIM Synchronization Input Source**

- `uint32_t HRTIM_InitTypeDef::SyncOutputSource`

Specifies the source and event to be sent on the external synchronization outputs This parameter can be a value of **HRTIM Synchronization Output Source**

- `uint32_t HRTIM_InitTypeDef::SyncOutputPolarity`

Specifies the conditionning of the event to be sent on the external synchronization outputs This parameter can be a value of **HRTIM Synchronization Output Polarity**

31.1.2 HRTIM_TimerParamTypeDef

Data Fields

- `uint32_t CaptureTrigger1`
- `uint32_t CaptureTrigger2`
- `uint32_t InterruptRequests`
- `uint32_t DMARequests`
- `uint32_t DMASrcAddress`
- `uint32_t DMADstAddress`
- `uint32_t DMASize`

Field Documentation

- `uint32_t HRTIM_TimerParamTypeDef::CaptureTrigger1`

Event(s) triggering capture unit 1. When the timer operates in Simple mode, this parameter can be a value of **HRTIM External Event Channels**. When the timer operates in Waveform mode, this parameter can be a combination of **HRTIM Capture Unit Trigger**.

- **`uint32_t HRTIM_TimerParamTypeDef::CaptureTrigger2`**
Event(s) triggering capture unit 2. When the timer operates in Simple mode, this parameter can be a value of **`HRTIM_External_Event_Channels`**. When the timer operates in Waveform mode, this parameter can be a combination of **`HRTIM_Capture_Unit_Trigger`**.
- **`uint32_t HRTIM_TimerParamTypeDef::InterruptRequests`**
Interrupts requests enabled for the timer.
- **`uint32_t HRTIM_TimerParamTypeDef::DMARequests`**
DMA requests enabled for the timer.
- **`uint32_t HRTIM_TimerParamTypeDef::DMASrcAddress`**
Address of the source address of the DMA transfer.
- **`uint32_t HRTIM_TimerParamTypeDef::DMADstAddress`**
Address of the destination address of the DMA transfer.
- **`uint32_t HRTIM_TimerParamTypeDef::DMASize`**
Size of the DMA transfer

31.1.3 **`__HRTIM_HandleTypeDef`**

Data Fields

- **`HRTIM_TypeDef * Instance`**
- **`HRTIM_InitTypeDef Init`**
- **`HRTIM_TimerParamTypeDef TimerParam`**
- **`HAL_LockTypeDef Lock`**
- **`_IO HAL_HRTIM_StateTypeDef State`**
- **`DMA_HandleTypeDef * hdmaMaster`**
- **`DMA_HandleTypeDef * hdmaTimerA`**
- **`DMA_HandleTypeDef * hdmaTimerB`**
- **`DMA_HandleTypeDef * hdmaTimerC`**
- **`DMA_HandleTypeDef * hdmaTimerD`**
- **`DMA_HandleTypeDef * hdmaTimerE`**

Field Documentation

- **`HRTIM_TypeDef* __HRTIM_HandleTypeDef::Instance`**
Register base address
- **`HRTIM_InitTypeDef __HRTIM_HandleTypeDef::Init`**
HRTIM required parameters
- **`HRTIM_TimerParamTypeDef __HRTIM_HandleTypeDef::TimerParam[MAX_HRTIM_TIMER]`**
HRTIM timers - including the master - parameters
- **`HAL_LockTypeDef __HRTIM_HandleTypeDef::Lock`**
Locking object
- **`_IO HAL_HRTIM_StateTypeDef __HRTIM_HandleTypeDef::State`**
HRTIM communication state
- **`DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaMaster`**
Master timer DMA handle parameters
- **`DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerA`**
Timer A DMA handle parameters

- **DMA_HandleTypeDef* __HRTIM_HandleTypeDefDef::hdmaTimerB**
Timer B DMA handle parameters
- **DMA_HandleTypeDef* __HRTIM_HandleTypeDefDef::hdmaTimerC**
Timer C DMA handle parameters
- **DMA_HandleTypeDef* __HRTIM_HandleTypeDefDef::hdmaTimerD**
Timer D DMA handle parameters
- **DMA_HandleTypeDef* __HRTIM_HandleTypeDefDef::hdmaTimerE**
Timer E DMA handle parameters

31.1.4 HRTIM_TimeBaseCfgTypeDef

Data Fields

- **uint32_t Period**
- **uint32_t RepetitionCounter**
- **uint32_t PrescalerRatio**
- **uint32_t Mode**

Field Documentation

- **uint32_t HRTIM_TimeBaseCfgTypeDef::Period**
Specifies the timer period The period value must be above 3 periods of the fHRTIM clock. Maximum value is = 0xFFDF
- **uint32_t HRTIM_TimeBaseCfgTypeDef::RepetitionCounter**
Specifies the timer repetition period This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- **uint32_t HRTIM_TimeBaseCfgTypeDef::PrescalerRatio**
Specifies the timer clock prescaler ratio. This parameter can be any value of **HRTIM Prescaler Ratio**
- **uint32_t HRTIM_TimeBaseCfgTypeDef::Mode**
Specifies the counter operating mode This parameter can be any value of **HRTIM Mode**

31.1.5 HRTIM_SimpleOCChannelCfgTypeDef

Data Fields

- **uint32_t Mode**
- **uint32_t Pulse**
- **uint32_t Polarity**
- **uint32_t IdleLevel**

Field Documentation

- **uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Mode**
Specifies the output compare mode (toggle, active, inactive) This parameter can be any value of **HRTIM Simple OC Mode**
- **uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Pulse**
Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- **uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Polarity**
Specifies the output polarity This parameter can be any value of **HRTIM Output Polarity**
- **uint32_t HRTIM_SimpleOCChannelCfgTypeDef::IdleLevel**

Specifies whether the output level is active or inactive when in IDLE state This parameter can be any value of **HRTIM Output IDLE Level**

31.1.6 HRTIM_SimplePWMChannelCfgTypeDef

Data Fields

- `uint32_t Pulse`
- `uint32_t Polarity`
- `uint32_t IdleLevel`

Field Documentation

- `uint32_t HRTIM_SimplePWMChannelCfgTypeDef::Pulse`

Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock

- `uint32_t HRTIM_SimplePWMChannelCfgTypeDef::Polarity`

Specifies the output polarity This parameter can be any value of **HRTIM Output Polarity**

- `uint32_t HRTIM_SimplePWMChannelCfgTypeDef::IdleLevel`

Specifies whether the output level is active or inactive when in IDLE state This parameter can be any value of **HRTIM Output IDLE Level**

31.1.7 HRTIM_SimpleCaptureChannelCfgTypeDef

Data Fields

- `uint32_t Event`
- `uint32_t EventPolarity`
- `uint32_t EventSensitivity`
- `uint32_t EventFilter`

Field Documentation

- `uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::Event`

Specifies the external event triggering the capture This parameter can be any 'EEVx' value of **HRTIM External Event Channels**

- `uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventPolarity`

Specifies the polarity of the external event (in case of level sensitivity) This parameter can be a value of **HRTIM External Event Polarity**

- `uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventSensitivity`

Specifies the sensitivity of the external event This parameter can be a value of **HRTIM External Event Sensitivity**

- `uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventFilter`

Defines the frequency used to sample the External Event and the length of the digital filter This parameter can be a value of **HRTIM External Event Filter**

31.1.8 HRTIM_SimpleOnePulseChannelCfgTypeDef

Data Fields

- `uint32_t Pulse`
- `uint32_t OutputPolarity`
- `uint32_t OutputIdleLevel`
- `uint32_t Event`
- `uint32_t EventPolarity`

- `uint32_t EventSensitivity`
- `uint32_t EventFilter`

Field Documentation

- `uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::Pulse`

Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock

- `uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::OutputPolarity`

Specifies the output polarity This parameter can be any value of **HRTIM Output Polarity**

- `uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::OutputIdleLevel`

Specifies whether the output level is active or inactive when in IDLE state This parameter can be any value of **HRTIM Output IDLE Level**

- `uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::Event`

Specifies the external event triggering the pulse generation This parameter can be any 'EEVx' value of **HRTIM External Event Channels**

- `uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventPolarity`

Specifies the polarity of the external event (in case of level sensitivity) This parameter can be a value of **HRTIM External Event Polarity**

- `uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventSensitivity`

Specifies the sensitivity of the external event This parameter can be a value of **HRTIM External Event Sensitivity**

- `uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventFilter`

Defines the frequency used to sample the External Event and the length of the digital filter This parameter can be a value of **HRTIM External Event Filter**

31.1.9 HRTIM_TimerCfgTypeDef

Data Fields

- `uint32_t InterruptRequests`
- `uint32_t DMARequests`
- `uint32_t DMASrcAddress`
- `uint32_t DMADstAddress`
- `uint32_t DMASize`
- `uint32_t HalfModeEnable`
- `uint32_t StartOnSync`
- `uint32_t ResetOnSync`
- `uint32_t DACSynchro`
- `uint32_t PreloadEnable`
- `uint32_t UpdateGating`
- `uint32_t BurstMode`
- `uint32_t RepetitionUpdate`
- `uint32_t PushPull`
- `uint32_t FaultEnable`
- `uint32_t FaultLock`
- `uint32_t DeadTimeInsertion`
- `uint32_t DelayedProtectionMode`
- `uint32_t UpdateTrigger`
- `uint32_t ResetTrigger`

- `uint32_t ResetUpdate`

Field Documentation

- `uint32_t HRTIM_TimerCfgTypeDef::InterruptRequests`

Relevant for all HRTIM timers, including the master Specifies which interrupts requests must be enabled for the timer This parameter can be any combination of **HRTIM Master Interrupt Enable** or **HRTIM_Timing_Unit_Interrupt_Enable**

- `uint32_t HRTIM_TimerCfgTypeDef::DMARequests`

Relevant for all HRTIM timers, including the master Specifies which DMA requests must be enabled for the timer This parameter can be any combination of **HRTIM Master DMA Request Enable** or **HRTIM_Timing_Unit_DMA_Request_Enable**

- `uint32_t HRTIM_TimerCfgTypeDef::DMASrcAddress`

Relevant for all HRTIM timers, including the master Specifies the address of the source address of the DMA transfer

- `uint32_t HRTIM_TimerCfgTypeDef::DMADstAddress`

Relevant for all HRTIM timers, including the master Specifies the address of the destination address of the DMA transfer

- `uint32_t HRTIM_TimerCfgTypeDef::DMASize`

Relevant for all HRTIM timers, including the master Specifies the size of the DMA transfer

- `uint32_t HRTIM_TimerCfgTypeDef::HalfModeEnable`

Relevant for all HRTIM timers, including the master Specifies whether or not half mode is enabled This parameter can be any value of **HRTIM Half Mode Enable**

- `uint32_t HRTIM_TimerCfgTypeDef::StartOnSync`

Relevant for all HRTIM timers, including the master Specifies whether or not timer is reset by a rising edge on the synchronization input (when enabled) This parameter can be any value of **HRTIM Start On Sync Input Event**

- `uint32_t HRTIM_TimerCfgTypeDef::ResetOnSync`

Relevant for all HRTIM timers, including the master Specifies whether or not timer is reset by a rising edge on the synchronization input (when enabled) This parameter can be any value of **HRTIM Reset On Sync Input Event**

- `uint32_t HRTIM_TimerCfgTypeDef::DACSynchro`

Relevant for all HRTIM timers, including the master Indicates whether or not a DAC synchronization event is generated This parameter can be any value of **HRTIM DAC Synchronization**

- `uint32_t HRTIM_TimerCfgTypeDef::PreloadEnable`

Relevant for all HRTIM timers, including the master Specifies whether or not register preload is enabled This parameter can be any value of **HRTIM Register Preload Enable**

- `uint32_t HRTIM_TimerCfgTypeDef::UpdateGating`

Relevant for all HRTIM timers, including the master Specifies how the update occurs with respect to a burst DMA transaction or update enable inputs (Slave timers only) This parameter can be any value of **HRTIM Update Gating**

- `uint32_t HRTIM_TimerCfgTypeDef::BurstMode`

Relevant for all HRTIM timers, including the master Specifies how the timer behaves during a burst mode operation This parameter can be any value of **HRTIM Timer Burst Mode**

- `uint32_t HRTIM_TimerCfgTypeDef::RepetitionUpdate`

Relevant for all HRTIM timers, including the master Specifies whether or not registers update is triggered by the repetition event This parameter can be any value of **HRTIM Timer Repetition Update**

- `uint32_t HRTIM_TimerCfgTypeDef::PushPull`

Relevant for Timer A to Timer E Specifies whether or not the push-pull mode is enabled This parameter can be any value of **HRTIM Timer Push Pull Mode**

- `uint32_t HRTIM_TimerCfgTypeDef::FaultEnable`

Relevant for Timer A to Timer E Specifies which fault channels are enabled for the timer This parameter can be a combination of **HRTIM Timer Fault Enabling**

- `uint32_t HRTIM_TimerCfgTypeDef::FaultLock`

Relevant for Timer A to Timer E Specifies whether or not fault enabling status is write protected This parameter can be a value of **HRTIM Timer Fault Lock**

- `uint32_t HRTIM_TimerCfgTypeDef::DeadTimeInsertion`

Relevant for Timer A to Timer E Specifies whether or not deadtime insertion is enabled for the timer This parameter can be a value of **HRTIM Timer Deadtime Insertion**

- `uint32_t HRTIM_TimerCfgTypeDef::DelayedProtectionMode`

Relevant for Timer A to Timer E Specifies the delayed protection mode This parameter can be a value of **HRTIM Timer Delayed Protection Mode**

- `uint32_t HRTIM_TimerCfgTypeDef::UpdateTrigger`

Relevant for Timer A to Timer E Specifies source(s) triggering the timer registers update This parameter can be a combination of **HRTIM Timer Update Trigger**

- `uint32_t HRTIM_TimerCfgTypeDef::ResetTrigger`

Relevant for Timer A to Timer E Specifies source(s) triggering the timer counter reset This parameter can be a combination of **HRTIM Timer Reset Trigger**

- `uint32_t HRTIM_TimerCfgTypeDef::ResetUpdate`

Relevant for Timer A to Timer E Specifies whether or not registers update is triggered when the timer counter is reset This parameter can be a value of **HRTIM Timer Reset Update**

31.1.10 HRTIM_CompareCfgTypeDef

Data Fields

- `uint32_t CompareValue`
- `uint32_t AutoDelayedMode`
- `uint32_t AutoDelayedTimeout`

Field Documentation

- `uint32_t HRTIM_CompareCfgTypeDef::CompareValue`

Specifies the compare value of the timer compare unit the minimum value must be greater than or equal to 3 periods of the fHRTIM clock the maximum value must be less than or equal to 0xFFFF - 1 periods of the fHRTIM clock

- `uint32_t HRTIM_CompareCfgTypeDef::AutoDelayedMode`

Specifies the auto delayed mode for compare unit 2 or 4 This parameter can be a value of **HRTIM Compare Unit Auto Delayed Mode**

- `uint32_t HRTIM_CompareCfgTypeDef::AutoDelayedTimeout`

Specifies compare value for timing unit 1 or 3 when auto delayed mode with time out is selected CompareValue + AutoDelayedTimeout must be less than 0xFFFF

31.1.11 HRTIM_CaptureCfgTypeDef

Data Fields

- `uint32_t Trigger`

Field Documentation

- `uint32_t HRTIM_CaptureCfgTypeDef::Trigger`

Specifies source(s) triggering the capture. This parameter can be a combination of **HRTIM Capture Unit Trigger**

31.1.12 HRTIM_OutputCfgTypeDef

Data Fields

- `uint32_t Polarity`
- `uint32_t SetSource`
- `uint32_t ResetSource`
- `uint32_t IdleMode`
- `uint32_t IdleLevel`
- `uint32_t FaultLevel`
- `uint32_t ChopperModeEnable`
- `uint32_t BurstModeEntryDelayed`

Field Documentation

- `uint32_t HRTIM_OutputCfgTypeDef::Polarity`

Specifies the output polarity. This parameter can be any value of **HRTIM Output Polarity**

- `uint32_t HRTIM_OutputCfgTypeDef::SetSource`

Specifies the event(s) transitioning the output from its inactive level to its active level. This parameter can be a combination of **HRTIM Output Set Source**

- `uint32_t HRTIM_OutputCfgTypeDef::ResetSource`

Specifies the event(s) transitioning the output from its active level to its inactive level. This parameter can be a combination of **HRTIM Output Reset Source**

- `uint32_t HRTIM_OutputCfgTypeDef::IdleMode`

Specifies whether or not the output is affected by a burst mode operation. This parameter can be any value of **HRTIM Output Idle Mode**

- `uint32_t HRTIM_OutputCfgTypeDef::IdleLevel`

Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of **HRTIM Output IDLE Level**

- `uint32_t HRTIM_OutputCfgTypeDef::FaultLevel`

Specifies whether the output level is active or inactive when in FAULT state. This parameter can be any value of **HRTIM Output FAULT Level**

- `uint32_t HRTIM_OutputCfgTypeDef::ChopperModeEnable`

Indicates whether or not the chopper mode is enabled. This parameter can be any value of **HRTIM Output Chopper Mode Enable**

- `uint32_t HRTIM_OutputCfgTypeDef::BurstModeEntryDelayed`

31.1.13 HRTIM_TimerEventFilteringCfgTypeDef

Data Fields

- `uint32_t Filter`
- `uint32_t Latch`

Field Documentation

- `uint32_t HRTIM_TimerEventFilteringCfgTypeDef::Filter`

Specifies the type of event filtering within the timing unit This parameter can be a value of **HRTIM Timer External Event Filter**

- **`uint32_t HRTIM_TimerEventFilteringCfgTypeDef::Latch`**

Specifies whether or not the signal is latched This parameter can be a value of **HRTIM Timer External Event Latch**

31.1.14 HRTIM_DeadTimeCfgTypeDef

Data Fields

- **`uint32_t Prescaler`**
- **`uint32_t RisingValue`**
- **`uint32_t RisingSign`**
- **`uint32_t RisingLock`**
- **`uint32_t RisingSignLock`**
- **`uint32_t FallingValue`**
- **`uint32_t FallingSign`**
- **`uint32_t FallingLock`**
- **`uint32_t FallingSignLock`**

Field Documentation

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::Prescaler`**

Specifies the Deadtime Prescaler This parameter can be a value of **HRTIM Deadtime Prescaler Ratio**

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::RisingValue`**

Specifies the Deadtime following a rising edge This parameter can be a number between 0x0 and 0x1FF

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::RisingSign`**

Specifies whether the deadtime is positive or negative on rising edge This parameter can be a value of **HRTIM Deadtime Rising Sign**

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::RisingLock`**

Specifies whether or not deadtime rising settings (value and sign) are write protected This parameter can be a value of **HRTIM Deadtime Rising Lock**

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::RisingSignLock`**

Specifies whether or not deadtime rising sign is write protected This parameter can be a value of **HRTIM Deadtime Rising Sign Lock**

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::FallingValue`**

Specifies the Deadtime following a falling edge This parameter can be a number between 0x0 and 0x1FF

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::FallingSign`**

Specifies whether the deadtime is positive or negative on falling edge This parameter can be a value of **HRTIM Deadtime Falling Sign**

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::FallingLock`**

Specifies whether or not deadtime falling settings (value and sign) are write protected This parameter can be a value of **HRTIM Deadtime Falling Lock**

- **`uint32_t HRTIM_DeadTimeCfgTypeDef::FallingSignLock`**

Specifies whether or not deadtime falling sign is write protected This parameter can be a value of **HRTIM Deadtime Falling Sign Lock**

31.1.15 HRTIM_ChopperModeCfgTypeDef

Data Fields

- `uint32_t CarrierFreq`
- `uint32_t DutyCycle`
- `uint32_t StartPulse`

Field Documentation

- `uint32_t HRTIM_ChopperModeCfgTypeDef::CarrierFreq`
Specifies the Timer carrier frequency value. This parameter can be a value of **HRTIM Chopper Frequency**
- `uint32_t HRTIM_ChopperModeCfgTypeDef::DutyCycle`
Specifies the Timer chopper duty cycle value. This parameter can be a value of **HRTIM Chopper Duty Cycle**
- `uint32_t HRTIM_ChopperModeCfgTypeDef::StartPulse`
Specifies the Timer pulse width value. This parameter can be a value of **HRTIM Chopper Start Pulse Width**

31.1.16 HRTIM_EventCfgTypeDef

Data Fields

- `uint32_t Source`
- `uint32_t Polarity`
- `uint32_t Sensitivity`
- `uint32_t Filter`
- `uint32_t FastMode`

Field Documentation

- `uint32_t HRTIM_EventCfgTypeDef::Source`
Identifies the source of the external event This parameter can be a value of **HRTIM External Event Sources**
- `uint32_t HRTIM_EventCfgTypeDef::Polarity`
Specifies the polarity of the external event (in case of level sensitivity) This parameter can be a value of **HRTIM External Event Polarity**
- `uint32_t HRTIM_EventCfgTypeDef::Sensitivity`
Specifies the sensitivity of the external event This parameter can be a value of **HRTIM External Event Sensitivity**
- `uint32_t HRTIM_EventCfgTypeDef::Filter`
Defines the frequency used to sample the External Event and the length of the digital filter This parameter can be a value of **HRTIM External Event Filter**
- `uint32_t HRTIM_EventCfgTypeDef::FastMode`
Indicates whether or not low latency mode is enabled for the external event This parameter can be a value of **HRTIM External Event Fast Mode**

31.1.17 HRTIM_FaultCfgTypeDef

Data Fields

- `uint32_t Source`
- `uint32_t Polarity`
- `uint32_t Filter`
- `uint32_t Lock`

Field Documentation

- `uint32_t HRTIM_FaultCfgTypeDef::Source`

Identifies the source of the fault This parameter can be a value of **HRTIM Fault Sources**

- **`uint32_t HRTIM_FaultCfgTypeDef::Polarity`**

Specifies the polarity of the fault event This parameter can be a value of **HRTIM Fault Polarity**

- **`uint32_t HRTIM_FaultCfgTypeDef::Filter`**

Defines the frequency used to sample the Fault input and the length of the digital filter This parameter can be a value of **HRTIM Fault Filter**

- **`uint32_t HRTIM_FaultCfgTypeDef::Lock`**

Indicates whether or not fault programming bits are write protected This parameter can be a value of **HRTIM Fault Lock**

31.1.18 HRTIM_BurstModeCfgTypeDef

Data Fields

- **`uint32_t Mode`**
- **`uint32_t ClockSource`**
- **`uint32_t Prescaler`**
- **`uint32_t PreloadEnable`**
- **`uint32_t Trigger`**
- **`uint32_t IdleDuration`**
- **`uint32_t Period`**

Field Documentation

- **`uint32_t HRTIM_BurstModeCfgTypeDef::Mode`**

Specifies the burst mode operating mode This parameter can be a value of **HRTIM Burst Mode Operating Mode**

- **`uint32_t HRTIM_BurstModeCfgTypeDef::ClockSource`**

Specifies the burst mode clock source This parameter can be a value of **HRTIM Burst Mode Clock Source**

- **`uint32_t HRTIM_BurstModeCfgTypeDef::Prescaler`**

Specifies the burst mode prescaler This parameter can be a value of **HRTIM Burst Mode Prescaler**

- **`uint32_t HRTIM_BurstModeCfgTypeDef::PreloadEnable`**

Specifies whether or not preload is enabled for burst mode related registers (HRTIM_BMCMPR and HRTIM_BMPER) This parameter can be a combination of **HRTIM Burst Mode Register Preload Enable**

- **`uint32_t HRTIM_BurstModeCfgTypeDef::Trigger`**

Specifies the event(s) triggering the burst operation This parameter can be a combination of **HRTIM Burst Mode Trigger**

- **`uint32_t HRTIM_BurstModeCfgTypeDef::IdleDuration`**

Specifies number of periods during which the selected timers are in idle state This parameter can be a number between 0x0 and 0xFFFF

- **`uint32_t HRTIM_BurstModeCfgTypeDef::Period`**

Specifies burst mode repetition period This parameter can be a number between 0x1 and 0xFFFF

31.1.19 HRTIM_ADCTriggerCfgTypeDef

Data Fields

- **`uint32_t UpdateSource`**
- **`uint32_t Trigger`**

Field Documentation

- **`uint32_t HRTIM_ADCTriggerCfgTypeDef::UpdateSource`**
Specifies the ADC trigger update source This parameter can be a combination of **HRTIM ADC Trigger Update Source**
- **`uint32_t HRTIM_ADCTriggerCfgTypeDef::Trigger`**
Specifies the event(s) triggering the ADC conversion This parameter can be a value of **HRTIM ADC Trigger Event**

31.2 HRTIM Firmware driver API description

31.2.1 Simple mode v.s. waveform mode

The HRTIM HAL API is split into 2 categories:

1. Simple functions: these functions allow for using a HRTIM timer as a general purpose timer with high resolution capabilities. Following simple modes are proposed: (+)Output compare mode (+)PWM output mode (+)Input capture mode (+)One pulse mode HRTIM simple modes are managed through the set of functions named `HAL_HRTIM_Simple<Function>`. These functions are similar in name and usage to the one defined for the TIM peripheral. When a HRTIM timer operates in simple mode, only a very limited set of HRTIM features are used.
2. Waveform functions: These functions allow taking advantage of the HRTIM flexibility to produce numerous types of control signal. When a HRTIM timer operates in waveform mode, all the HRTIM features are accessible without any restriction. HRTIM waveform modes are managed through the set of functions named `HAL_HRTIM_Waveform<Function>`

The HRTIM HAL API is split into 2 categories: (#)Simple functions: these functions allow for using a HRTIM timer as a general purpose timer with high resolution capabilities. Following simple modes are proposed:

- Output compare mode
- PWM output mode
- Input capture mode
- One pulse mode HRTIM simple modes are managed through the set of functions named `HAL_HRTIM_Simple<Function>`. These functions are similar in name and usage to the one defined for the TIM peripheral. When a HRTIM timer operates in simple mode, only a very limited set of HRTIM features are used. (#)Waveform functions: These functions allow taking advantage of the HRTIM flexibility to produce numerous types of control signal. When a HRTIM timer operates in waveform mode, all the HRTIM features are accessible without any restriction. HRTIM waveform modes are managed through the set of functions named `HAL_HRTIM_Waveform<Function>`

31.2.2 How to use this driver

1. Initialize the HRTIM low level resources by implementing the `HAL_HRTIM_MspInit()` function:
 - Enable the HRTIM clock source using `__HRTIMx_CLK_ENABLE()`
 - Connect HRTIM pins to MCU IOs
 - Enable the clock for the HRTIM GPIOs using the following function: `__GPIOx_CLK_ENABLE()`
 - Configure these GPIO pins in Alternate Function mode using `HAL_GPIO_Init()`
 - When using DMA to control data transfer (e.g `HAL_HRTIM_SimpleBaseStart_DMA()`)
 - Enable the DMAx interface clock using `__DMAx_CLK_ENABLE()`
 - Initialize the DMA handle
 - Associate the initialized DMA handle to the appropriate DMA handle of the HRTIM handle using `__HAL_LINKDMA()`
 - Initialize the DMA channel using `HAL_DMA_Init()`
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA channel using `HAL_NVIC_SetPriority()` and `HAL_NVIC_EnableIRQ()`
 - In case of using interrupt mode (e.g `HAL_HRTIM_SimpleBaseStart_IT()`)

- Configure the priority and enable the NVIC for the concerned HRTIM interrupt using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HRTIM HAL using HAL_HRTIM_Init(). The HRTIM configuration structure (field of the HRTIM handle) specifies which global interrupt of whole HRTIM must be enabled (Burst mode period, System fault, Faults). It also contains the HRTIM external synchronization configuration. HRTIM can act as a master (generating a synchronization signal) or as a slave (waiting for a trigger to be synchronized).
3. Configure HRTIM resources shared by all HRTIM timers
- Burst Mode Controller:
 - HAL_HRTIM_BurstModeConfig(): configures the HRTIM burst mode controller: operating mode (continuous or shot mode), clock (source, prescaler) , trigger(s), period, idle duration.
 - External Events Conditionning:
 - HAL_HRTIM_EventConfig(): configures the conditioning of an external event channel: source, polarity, edge sensitivity. External event can be used as triggers (timer reset, input capture, burst mode, ADC triggers, delayed protection,,,) They can also be used to set or reset timer outputs. Up to 10 event channels are available.
 - HAL_HRTIM_EventPrescalerConfig(): configures the external event sampling clock (used for digital filtering).
 - Fault Conditionning:
 - HAL_HRTIM_FaultConfig(): configures the conditioning of a fault channel: source, polarity, edge sensitivity. Fault channels are used to disable the outputs in case of an abnormal operation. Up to 5 fault channels are available.
 - HAL_HRTIM_FaultPrescalerConfig(): configures the fault sampling clock (used for digital filtering).
 - HAL_HRTIM_FaultModeCtl(): Enables or disables fault input(s) circuitry. By default all fault inputs are disabled.
 - ADC trigger:
 - HAL_HRTIM_ADCTriggerConfig(): configures the source triggering the update of the ADC trigger register and the ADC trigger. 4 independent triggers are available to start both the regular and the injected sequencers of the 2 ADCs.
4. Configure HRTIM timer time base using HAL_HRTIM_TimeBaseConfig(). This function must be called whatever the HRTIM timer operating mode is (simple v.s. waveform). It configures mainly:
- a. The HRTIM timer counter operating mode (continuous, one shot)
 - b. The HRTIM timer clock prescaler
 - c. The HRTIM timer period
 - d. The HRTIM timer repetition counter
5. If the HRTIM timer operates in simple mode:
- a. Simple time base: HAL_HRTIM_SimpleBaseStart(),HAL_HRTIM_SimpleBaseStop(),
HAL_HRTIM_SimpleBaseStart_IT(),HAL_HRTIM_SimpleBaseStop_IT(),
HAL_HRTIM_SimpleBaseStart_DMA(),HAL_HRTIM_SimpleBaseStop_DMA().
 - b. Simple output compare: HAL_HRTIM_SimpleOCChannelConfig(),
HAL_HRTIM_SimpleOCStart(),HAL_HRTIM_SimpleOCStop(),
HAL_HRTIM_SimpleOCStart_IT(),HAL_HRTIM_SimpleOCStop_IT(),
HAL_HRTIM_SimpleOCStart_DMA(),HAL_HRTIM_SimpleOCStop_DMA(),
 - c. Simple PWM output: HAL_HRTIM_SimplePWMChannelConfig(),
HAL_HRTIM_SimplePWMStart(),HAL_HRTIM_SimplePWMStop(),
HAL_HRTIM_SimplePWMStart_IT(),HAL_HRTIM_SimplePWMStop_IT(),
HAL_HRTIM_SimplePWMStart_DMA(),HAL_HRTIM_SimplePWMStop_DMA(),
 - d. Simple input capture: HAL_HRTIM_SimpleCaptureChannelConfig(),
HAL_HRTIM_SimpleCaptureStart(),HAL_HRTIM_SimpleCaptureStop(),
HAL_HRTIM_SimpleCaptureStart_IT(),HAL_HRTIM_SimpleCaptureStop_IT(),
HAL_HRTIM_SimpleCaptureStart_DMA(),HAL_HRTIM_SimpleCaptureStop_DMA(),
 - e. Simple one pulse: HAL_HRTIM_SimpleOnePulseChannelConfig(),
HAL_HRTIM_SimpleOnePulseStart(),HAL_HRTIM_SimpleOnePulseStop(),
HAL_HRTIM_SimpleOnePulseStart_IT(),HAL_HRTIM_SimpleOnePulseStop_IT().
6. If the HRTIM timer operates in waveform mode:
- a. Completes waveform timer configuration

- HAL_HRTIM_WaveformTimerConfig(): configuration of a HRTIM timer operating in wave form mode mainly consists in:
 - Enabling the HRTIM timer interrupts and DMA requests,
 - Enabling the half mode for the HRTIM timer,
 - Defining how the HRTIM timer reacts to external synchronization input,
 - Enabling the push pull mode for the HRTIM timer,
 - Enabling the fault channels for the HRTIM timer,
 - Enabling the deadtime insertion for the HRTIM timer,
 - Setting the delayed protection mode for the HRTIM timer (source and outputs on which the delayed protection are applied),
 - Specifying the HRTIM timer update and reset triggers,
 - Specifying the HRTIM timer registers update policy (preload enabling...).
 - HAL_HRTIM_TimerEventFilteringConfig(): configures external event blanking and windowingcircuitry of a HRTIM timer:
 - Blanking: to mask external events during a defined time period
 - Windowing: to enable external events only during a defined time period
 - HAL_HRTIM_DeadTimeConfig(): configures the deadtime insertion unit for a HRTIM timer. Allows to generate a couple of complementary signals from a single reference waveform, with programmable delays between active state.
 - HAL_HRTIM_ChopperModeConfig(): configures the parameters of the high frequency carrier signal added on top of the timing unit output. Chopper mode can be enabled or disabled for each timer output separately (see HAL_HRTIM_WaveformOutputConfig()).
 - HAL_HRTIM_BurstDMAConfig(): configures the burst DMA burst controller. Allows having multiple HRTIM registers updated with a single DMA request. The burst DMA operation is started by calling HAL_HRTIM_BurstDMATransfer().
 - HAL_HRTIM_WaveformCompareConfig(): configures the compare unit of a HRTIM timer. This operation consists in setting the compare value and possibly specifying the auto delayed mode for compare units 2 and 4 (allows to have compare events generated relatively to capture events). Note that when auto delayed mode is needed, the capture unit associated to the compare unit must be configured separately.
 - HAL_HRTIM_WaveformCaptureConfig(): configures the capture unit of a HRTIM timer. This operation consists in specifying the source(s) triggering the capture (timer register update event, external event, timer output set or reset event, other HRTIM timer related events).
 - HAL_HRTIM_WaveformOutputConfig(): configuration HRTIM timer output manly consists in:
 - Setting the output polarity (active high or active low),
 - Defining the set or reset crossbar for the output,
 - Specifying the fault level (active or inactive) in IDLE and FAULT states.
- b. Set waveform timer output(s) level
- HAL_HRTIM_WaveformSetOutputLevel(): forces the output to its active or inactive level. For example, when deadtime insertion is enabled it is necessary to force the output level by software to have the outputs in a complementary state as soon as the RUN mode is entered.
- c. Enable or Disable waveform timer output(s)
- HAL_HRTIM_WaveformOutputStart(),HAL_HRTIM_WaveformOutputStop().
- d. Start or Stop waveform HRTIM timer(s).
- HAL_HRTIM_WaveformCounterStart(),HAL_HRTIM_WaveformCounterStop(),
 - HAL_HRTIM_WaveformCounterStart_IT(),HAL_HRTIM_WaveformCounterStop_IT(),
 - HAL_HRTIM_WaveformCounterStart()_DMA,HAL_HRTIM_WaveformCounterStop_DMA(),
- e. Burst mode controller enabling:
- HAL_HRTIM_BurstModeCtl(): activates or deactivates the burst mode controller.
- f. Some HRTIM operations can be triggered by software:
- HAL_HRTIM_BurstModeSoftwareTrigger(): calling this function trigs the burst operation.
 - HAL_HRTIM_SoftwareCapture(): calling this function trigs the capture of the HRTIM timer counter.

- HAL_HRTIM_SoftwareUpdate(): calling this function trigs the update of the preloadable registers of the HRTIM timer ()
 - HAL_HRTIM_SoftwareReset(): calling this function resets the HRTIM timer counter.
- g. Some functions can be used anytime to retrieve HRTIM timer related information
- HAL_HRTIM_GetCapturedValue(): returns actual value of the capture register of the designated capture unit.
 - HAL_HRTIM_WaveformGetOutputLevel(): returns actual level (ACTIVE or INACTIVE) of the designated timer output.
 - HAL_HRTIM_WaveformGetOutputState(): returns actual state (IDLE, RUN or FAULT) of the designated timer output.
 - HAL_HRTIM_GetDelayedProtectionStatus(): returns actual level (ACTIVE or INACTIVE) of the designated output when the delayed protection was triggered.
 - HAL_HRTIM_GetBurstStatus(): returns the actual status (ACTIVE or INACTIVE) of the burst mode controller.
 - HAL_HRTIM_GetCurrentPushPullStatus(): when the push pull mode is enabled for the HRTIM timer (see HAL_HRTIM_WaveformTimerConfig()), the push pull indicates on which output the signal is currently active (e.g. signal applied on output 1 and output 2 forced inactive or vice versa).
 - HAL_HRTIM_GetIdlePushPullStatus(): when the push pull mode is enabled for the HRTIM timer (see HAL_HRTIM_WaveformTimerConfig()), the idle push pull status indicates during which period the delayed protection request occurred (e.g. protection occurred when the output 1 was active and output 2 forced inactive or vice versa).
- h. Some functions can be used anytime to retrieve actual HRTIM status
- HAL_HRTIM_GetState(): returns actual HRTIM instance HAL state.

31.2.3

Initialization and Time Base Configuration functions

This section provides functions allowing to:

- Initialize a HRTIM instance
- De-initialize a HRTIM instance
- Initialize the HRTIM MSP
- De-initialize the HRTIM MSP
- Configure the time base unit of a HRTIM timer

This section contains the following APIs:

- [**HAL_HRTIM_Init**](#)
- [**HAL_HRTIM_DelInit**](#)
- [**HAL_HRTIM_MspInit**](#)
- [**HAL_HRTIM_MspDelInit**](#)
- [**HAL_HRTIM_TimeBaseConfig**](#)

31.2.4

Simple time base mode functions

This section provides functions allowing to:

- Start simple time base
- Stop simple time base
- Start simple time base and enable interrupt
- Stop simple time base and disable interrupt
- Start simple time base and enable DMA transfer
- Stop simple time base and disable DMA transfer

Note:

When a HRTIM timer operates in simple time base mode, the timer counter counts from 0 to the period value.

This section contains the following APIs:

- [`HAL_HRTIM_SimpleBaseStart`](#)
- [`HAL_HRTIM_SimpleBaseStop`](#)
- [`HAL_HRTIM_SimpleBaseStart_IT`](#)
- [`HAL_HRTIM_SimpleBaseStop_IT`](#)
- [`HAL_HRTIM_SimpleBaseStart_DMA`](#)
- [`HAL_HRTIM_SimpleBaseStop_DMA`](#)

31.2.5 Simple output compare functions

This section provides functions allowing to:

- Configure simple output channel
- Start simple output compare
- Stop simple output compare
- Start simple output compare and enable interrupt
- Stop simple output compare and disable interrupt
- Start simple output compare and enable DMA transfer
- Stop simple output compare and disable DMA transfer

Note:

When a HRTIM timer operates in simple output compare mode the output level is set to a programmable value when a match is found between the compare register and the counter. Compare unit 1 is automatically associated to output 1 Compare unit 2 is automatically associated to output 2

This section contains the following APIs:

- [`HAL_HRTIM_SimpleOCChannelConfig`](#)
- [`HAL_HRTIM_SimpleOCStart`](#)
- [`HAL_HRTIM_SimpleOCStop`](#)
- [`HAL_HRTIM_SimpleOCStart_IT`](#)
- [`HAL_HRTIM_SimpleOCStop_IT`](#)
- [`HAL_HRTIM_SimpleOCStart_DMA`](#)
- [`HAL_HRTIM_SimpleOCStop_DMA`](#)

31.2.6 Simple PWM output functions

This section provides functions allowing to:

- Configure simple PWM output channel
- Start simple PWM output
- Stop simple PWM output
- Start simple PWM output and enable interrupt
- Stop simple PWM output and disable interrupt
- Start simple PWM output and enable DMA transfer
- Stop simple PWM output and disable DMA transfer

Note:

When a HRTIM timer operates in simple PWM output mode the output level is set to a programmable value when a match is found between the compare register and the counter and reset when the timer period is reached. Duty cycle is determined by the comparison value. Compare unit 1 is automatically associated to output 1 Compare unit 2 is automatically associated to output 2

This section contains the following APIs:

- [`HAL_HRTIM_SimplePWMChannelConfig`](#)
- [`HAL_HRTIM_SimplePWMStart`](#)
- [`HAL_HRTIM_SimplePWMStop`](#)
- [`HAL_HRTIM_SimplePWMStart_IT`](#)
- [`HAL_HRTIM_SimplePWMStop_IT`](#)

- [*HAL_HRTIM_SimplePWMStart_DMA*](#)
- [*HAL_HRTIM_SimplePWMStop_DMA*](#)

31.2.7 Simple input capture functions

This section provides functions allowing to:

- Configure simple input capture channel
- Start simple input capture
- Stop simple input capture
- Start simple input capture and enable interrupt
- Stop simple input capture and disable interrupt
- Start simple input capture and enable DMA transfer
- Stop simple input capture and disable DMA transfer

Note: When a HRTIM timer operates in simple input capture mode the Capture Register (HRTIM_CPT1/2xR) is used to latch the value of the timer counter counter after a transition detected on a given external event input.

This section contains the following APIs:

- [*HAL_HRTIM_SimpleCaptureChannelConfig*](#)
- [*HAL_HRTIM_SimpleCaptureStart*](#)
- [*HAL_HRTIM_SimpleCaptureStop*](#)
- [*HAL_HRTIM_SimpleCaptureStart_IT*](#)
- [*HAL_HRTIM_SimpleCaptureStop_IT*](#)
- [*HAL_HRTIM_SimpleCaptureStart_DMA*](#)
- [*HAL_HRTIM_SimpleCaptureStop_DMA*](#)

31.2.8 Simple one pulse functions

This section provides functions allowing to:

- Configure one pulse channel
- Start one pulse generation
- Stop one pulse generation
- Start one pulse generation and enable interrupt
- Stop one pulse generation and disable interrupt

Note: When a HRTIM timer operates in simple one pulse mode the timer counter is started in response to transition detected on a given external event input to generate a pulse with a programmable length after a programmable delay.

This section contains the following APIs:

- [*HAL_HRTIM_SimpleOnePulseChannelConfig*](#)
- [*HAL_HRTIM_SimpleOnePulseStart*](#)
- [*HAL_HRTIM_SimpleOnePulseStop*](#)
- [*HAL_HRTIM_SimpleOnePulseStart_IT*](#)
- [*HAL_HRTIM_SimpleOnePulseStop_IT*](#)

31.2.9 HRTIM configuration functions

This section provides functions allowing to configure the HRTIM resources shared by all the HRTIM timers operating in waveform mode:

- Configure the burst mode controller
- Configure an external event conditionning
- Configure the external events sampling clock
- Configure a fault conditionning

- Enable or disable fault inputs
- Configure the faults sampling clock
- Configure an ADC trigger

This section contains the following APIs:

- [*HAL_HRTIM_BurstModeConfig*](#)
- [*HAL_HRTIM_EventConfig*](#)
- [*HAL_HRTIM_EventPrescalerConfig*](#)
- [*HAL_HRTIM_FaultConfig*](#)
- [*HAL_HRTIM_FaultPrescalerConfig*](#)
- [*HAL_HRTIM_FaultModeCtl*](#)
- [*HAL_HRTIM_ADCTriggerConfig*](#)

31.2.10 HRTIM timer configuration and control functions

This section provides functions used to configure and control a HRTIM timer operating in waveform mode:

- Configure HRTIM timer general behavior
- Configure HRTIM timer event filtering
- Configure HRTIM timer deadtime insertion
- Configure HRTIM timer chopper mode
- Configure HRTIM timer burst DMA
- Configure HRTIM timer compare unit
- Configure HRTIM timer capture unit
- Configure HRTIM timer output
- Set HRTIM timer output level
- Enable HRTIM timer output
- Disable HRTIM timer output
- Start HRTIM timer
- Stop HRTIM timer
- Start HRTIM timer and enable interrupt
- Stop HRTIM timer and disable interrupt
- Start HRTIM timer and enable DMA transfer
- Stop HRTIM timer and disable DMA transfer
- Enable or disable the burst mode controller
- Start the burst mode controller (by software)
- Trigger a Capture (by software)
- Update the HRTIM timer preloadable registers (by software)
- Reset the HRTIM timer counter (by software)
- Start a burst DMA transfer
- Enable timer register update
- Disable timer register update

This section contains the following APIs:

- [*HAL_HRTIM_WaveformTimerConfig*](#)
- [*HAL_HRTIM_TimerEventFilteringConfig*](#)
- [*HAL_HRTIM_DeadTimeConfig*](#)
- [*HAL_HRTIM_ChopperModeConfig*](#)
- [*HAL_HRTIM_BurstDMAConfig*](#)
- [*HAL_HRTIM_WaveformCompareConfig*](#)
- [*HAL_HRTIM_WaveformCaptureConfig*](#)

- [`HAL_HRTIM_WaveformOutputConfig`](#)
- [`HAL_HRTIM_WaveformSetOutputLevel`](#)
- [`HAL_HRTIM_WaveformOutputStart`](#)
- [`HAL_HRTIM_WaveformOutputStop`](#)
- [`HAL_HRTIM_WaveformCounterStart`](#)
- [`HAL_HRTIM_WaveformCounterStop`](#)
- [`HAL_HRTIM_WaveformCounterStart_IT`](#)
- [`HAL_HRTIM_WaveformCounterStop_IT`](#)
- [`HAL_HRTIM_WaveformCounterStart_DMA`](#)
- [`HAL_HRTIM_WaveformCounterStop_DMA`](#)
- [`HAL_HRTIM_BurstModeCtl`](#)
- [`HAL_HRTIM_BurstModeSoftwareTrigger`](#)
- [`HAL_HRTIM_SoftwareCapture`](#)
- [`HAL_HRTIM_SoftwareUpdate`](#)
- [`HAL_HRTIM_SoftwareReset`](#)
- [`HAL_HRTIM_BurstDMATransfer`](#)
- [`HAL_HRTIM_UpdateEnable`](#)
- [`HAL_HRTIM_UpdateDisable`](#)

31.2.11 Peripheral State functions

This section provides functions used to get HRTIM or HRTIM timer specific information:

- Get HRTIM HAL state
- Get captured value
- Get HRTIM timer output level
- Get HRTIM timer output state
- Get delayed protection status
- Get burst status
- Get current push-pull status
- Get idle push-pull status

This section contains the following APIs:

- [`HAL_HRTIM_GetState`](#)
- [`HAL_HRTIM_GetCapturedValue`](#)
- [`HAL_HRTIM_WaveformGetOutputLevel`](#)
- [`HAL_HRTIM_WaveformGetOutputState`](#)
- [`HAL_HRTIM_GetDelayedProtectionStatus`](#)
- [`HAL_HRTIM_GetBurstStatus`](#)
- [`HAL_HRTIM_GetCurrentPushPullStatus`](#)
- [`HAL_HRTIM_GetIdlePushPullStatus`](#)

31.2.12 Detailed description of functions

`HAL_HRTIM_Init`

Function name

`HAL_StatusTypeDef HAL_HRTIM_Init (HRTIM_HandleTypeDef * hhrtim)`

Function description

Initializes a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_DelInit

Function name

HAL_StatusTypeDef HAL_HRTIM_DelInit (HRTIM_HandleTypeDef * hhrtim)

Function description

De-initializes a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_MspInit

Function name

void HAL_HRTIM_MspInit (HRTIM_HandleTypeDef * hhrtim)

Function description

MSP initialization for a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_MspDelInit

Function name

void HAL_HRTIM_MspDelInit (HRTIM_HandleTypeDef * hhrtim)

Function description

MSP initialization for a for a HRTIM instance.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_TimeBaseConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_TimeBaseConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_TimeBaseCfgTypeDef * pTimeBaseCfg)

Function description

Configures the time base unit of a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **pTimeBaseCfg:** pointer to the time base configuration structure

Return values

- **HAL:** status

Notes

- This function must be called prior starting the timer
- The time-base unit initialization parameters specify: The timer counter operating mode (continuous, one shot), The timer clock prescaler, The timer period, The timer repetition counter.

HAL_HRTIM_SimpleBaseStart

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Starts the counter of a timer operating in basic time base mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStop

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Stops the counter of a timer operating in basic time base mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B

- HRTIM_TIMERINDEX_TIMER_C for timer C
- HRTIM_TIMERINDEX_TIMER_D for timer D
- HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStart_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Starts the counter of a timer operating in simple time base mode (Timer repetition interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Stops the counter of a timer operating in simple time base mode (Timer repetition interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleBaseStart_DMA

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)
```

Function description

Starts the counter of a timer operating in simple time base mode (Timer repetition DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

HAL_HRTIM_SimpleBaseStop_DMA

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx)
```

Function description

Stops the counter of a timer operating in simple time base mode (Timer repetition DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCChannelConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleOCChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t OCChannel, HRTIM_SimpleOCChannelCfgTypeDef * pSimpleOCChannelCfg)
```

Function description

Configures an output in simple output compare mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pSimpleOCChannelCfg:** pointer to the simple output compare output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple output compare mode: Output 1 is implicitly controlled by the compare unit 1 Output 2 is implicitly controlled by the compare unit 2 Output Set/Reset crossbar is set according to the selected output compare mode: Toggle: SETxyR = RSTxyR = CMPy Active: SETxyR = CMPy, RSTxyR = 0 Inactive: SETxy = 0, RSTxy = CMPy

HAL_HRTIM_SimpleOCStart

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,  
uint32_t OCChannel)
```

Function description

Starts the output compare signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2

- HRTIM_OUTPUT_TC1: Timer C - Output 1
- HRTIM_OUTPUT_TC2: Timer C - Output 2
- HRTIM_OUTPUT_TD1: Timer D - Output 1
- HRTIM_OUTPUT_TD2: Timer D - Output 2
- HRTIM_OUTPUT_TE1: Timer E - Output 1
- HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStop

Function name

**HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,
uint32_t OCChannel)**

Function description

Stops the output compare signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStart_IT

Function name

**HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,
uint32_t OCChannel)**

Function description

Starts the output compare signal generation on the designed timer output (Interrupt is enabled (see note note below)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

Notes

- Interrupt enabling depends on the chosen output compare mode Output toggle: compare match interrupt is enabled Output set active: output set interrupt is enabled Output set inactive: output reset interrupt is enabled

HAL_HRTIM_SimpleOCStop_IT

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,  
uint32_t OCChannel)
```

Function description

Stops the output compare signal generation on the designed timer output (Interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2

- HRTIM_OUTPUT_TE1: Timer E - Output 1
- HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOCStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Starts the output compare signal generation on the designed timer output (DMA request is enabled (see note below)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

Notes

- DMA request enabling depends on the chosen output compare mode Output toggle: compare match DMA request is enabled Output set active: output set DMA request is enabled Output set inactive: output reset DMA request is enabled

HAL_HRTIM_SimpleOCStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function description

Stops the output compare signal generation on the designed timer output (DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMChannelConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimplePWMChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t PWMChannel, HRTIM_SimplePWMChannelCfgTypeDef * pSimplePWMChannelCfg)
```

Function description

Configures an output in simple PWM mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1

- HRTIM_OUTPUT_TD2: Timer D - Output 2
- HRTIM_OUTPUT_TE1: Timer E - Output 1
- HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pSimplePWMChannelCfg:** pointer to the simple PWM output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple PWM output mode: Output 1 is implicitly controlled by the compare unit 1 Output 2 is implicitly controlled by the compare unit 2 Output Set/Reset crossbar is set as follows: Output 1: SETx1R = CMP1, RSTx1R = PER Output 2: SETx2R = CMP2, RST2R = PER
- When Simple PWM mode is used the registers preload mechanism is enabled (otherwise the behavior is not guaranteed).

HAL_HRTIM_SimplePWMStart

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,  
uint32_t PWMChannel)
```

Function description

Starts the PWM output signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMSStop

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimplePWMSStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,  
uint32_t PWMChannel)
```

Function description

Stops the PWM output signal generation on the designed timer output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStart_IT

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t PWMChannel)
```

Function description

Starts the PWM output signal generation on the designed timer output (The compare interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1

- HRTIM_OUTPUT_TD2: Timer D - Output 2
- HRTIM_OUTPUT_TE1: Timer E - Output 1
- HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Stops the PWM output signal generation on the designed timer output (The compare interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStart_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)

Function description

Starts the PWM output signal generation on the designed timer output (The compare DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B

- HRTIM_TIMERINDEX_TIMER_C for timer C
- HRTIM_TIMERINDEX_TIMER_D for timer D
- HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

HAL_HRTIM_SimplePWMStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function description

Stops the PWM output signal generation on the designed timer output (The compare DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureChannelConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureChannelConfig (HRTIM_HandleTypeDef * hhrtim,  
uint32_t TimerIdx, uint32_t CaptureChannel, HRTIM_SimpleCaptureChannelCfgTypeDef *  
pSimpleCaptureChannelCfg)
```

Function description

Configures a simple capture.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Capture unit This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **pSimpleCaptureChannelCfg:** pointer to the simple capture configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in simple capture mode the capture is triggered by the designated external event and GPIO input is implicitly used as event source. The capture can be triggered by a rising edge, a falling edge or both edges on event channel.

HAL_HRTIM_SimpleCaptureStart

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t CaptureChannel)
```

Function description

Enables a simple capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

Notes

- The external event triggering the capture is available for all timing units. It can be used directly and is active as soon as the timing unit counter is enabled.

HAL_HRTIM_SimpleCaptureStop

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t CaptureChannel)
```

Function description

Disables a simple capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStart_IT

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t CaptureChannel)
```

Function description

Enables a basic capture on the designed capture unit (Capture interrupt is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStop_IT

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)
```

Function description

Disables a basic capture on the designed capture unit (Capture interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStart_DMA

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)
```

Function description

Enables a basic capture on the designed capture unit (Capture DMA request is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values

- **HAL:** status

HAL_HRTIM_SimpleCaptureStop_DMA

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)
```

Function description

Disables a basic capture on the designed capture unit (Capture DMA request is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseChannelConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel, HRTIM_SimpleOnePulseChannelCfgTypeDef * pSimpleOnePulseChannelCfg)
```

Function description

Configures an output simple one pulse mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1

- HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pSimpleOnePulseChannelCfg:** pointer to the basic one pulse output configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in basic one pulse mode: the timer counter is implicitly started by the reset event, the reset of the timer counter is triggered by the designated external event GPIO input is implicitly used as event source, Output 1 is implicitly controlled by the compare unit 1, Output 2 is implicitly controlled by the compare unit 2. Output Set/Reset crossbar is set as follows: Output 1: SETx1R = CMP1, RSTx1R = PER Output 2: SETx2R = CMP2, RST2R = PER
- If HAL_HRTIM_SimpleOnePulseChannelConfig is called for both timer outputs, the reset event related configuration data provided in the second call will override the reset event related configuration data provided in the first call.

HAL_HRTIM_SimpleOnePulseStart

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStart (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t OnePulseChannel)
```

Function description

Enables the simple one pulse signal generation on the designed output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStop

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStop (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t OnePulseChannel)
```

Function description

Disables the simple one pulse signal generation on the designed output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStart_IT

Function name

```
HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t
TimerIdx, uint32_t OnePulseChannel)
```

Function description

Enables the simple one pulse signal generation on the designed output (The compare interrupt is enabled (pulse start)).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1

- HRTIM_OUTPUT_TD2: Timer D - Output 2
- HRTIM_OUTPUT_TE1: Timer E - Output 1
- HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_SimpleOnePulseStop_IT

Function name

HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)

Function description

Disables the simple one pulse signal generation on the designed output (The compare interrupt is disabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_BurstModeConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeConfig (HRTIM_HandleTypeDef * hhrtim, HRTIM_BurstModeCfgTypeDef * pBurstModeCfg)

Function description

Configures the burst mode feature of the HRTIM.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **pBurstModeCfg:** pointer to the burst mode configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the burst mode controller

HAL_HRTIM_EventConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_EventConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Event,  
HRTIM_EventCfgTypeDef * pEventCfg)
```

Function description

Configures the conditioning of an external event.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Event:** external event to configure This parameter can be one of the following values:
 - HRTIM_EVENT_1: External event 1
 - HRTIM_EVENT_2: External event 2
 - HRTIM_EVENT_3: External event 3
 - HRTIM_EVENT_4: External event 4
 - HRTIM_EVENT_5: External event 5
 - HRTIM_EVENT_6: External event 6
 - HRTIM_EVENT_7: External event 7
 - HRTIM_EVENT_8: External event 8
 - HRTIM_EVENT_9: External event 9
 - HRTIM_EVENT_10: External event 10
- **pEventCfg:** pointer to the event conditioning configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_EventPrescalerConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_EventPrescalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t  
Prescaler)
```

Function description

Configures the external event conditioning block prescaler.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Prescaler:** Prescaler value This parameter can be one of the following values:
 - HRTIM_EVENTPRESCALER_DIV1: fEEVS=fHRTIM
 - HRTIM_EVENTPRESCALER_DIV2: fEEVS=fHRTIM / 2
 - HRTIM_EVENTPRESCALER_DIV4: fEEVS=fHRTIM / 4
 - HRTIM_EVENTPRESCALER_DIV8: fEEVS=fHRTIM / 8

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_FaultConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_FaultConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Fault,  
HRTIM_FaultCfgTypeDef * pFaultCfg)
```

Function description

Configures the conditioning of fault input.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Fault:** fault input to configure This parameter can be one of the following values:
 - HRTIM_FAULT_1: Fault input 1
 - HRTIM_FAULT_2: Fault input 2
 - HRTIM_FAULT_3: Fault input 3
 - HRTIM_FAULT_4: Fault input 4
 - HRTIM_FAULT_5: Fault input 5
- **pFaultCfg:** pointer to the fault conditioning configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer and before enabling faults inputs

HAL_HRTIM_FaultPrescalerConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_FaultPrescalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t  
Prescaler)
```

Function description

Configures the fault conditioning block prescaler.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Prescaler:** Prescaler value This parameter can be one of the following values:
 - HRTIM_FAULTPRESCALER_DIV1: fFLTS=fHRTIM
 - HRTIM_FAULTPRESCALER_DIV2: fFLTS=fHRTIM / 2
 - HRTIM_FAULTPRESCALER_DIV4: fFLTS=fHRTIM / 4
 - HRTIM_FAULTPRESCALER_DIV8: fFLTS=fHRTIM / 8

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer and before enabling faults inputs

HAL_HRTIM_FaultModeCtl

Function name

```
void HAL_HRTIM_FaultModeCtl (HRTIM_HandleTypeDef * hhrtim, uint32_t Faults, uint32_t Enable)
```

Function description

Enables or disables the HRTIMx Fault mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Faults:** fault input(s) to enable or disable This parameter can be any combination of the following values:
 - HRTIM_FAULT_1: Fault input 1
 - HRTIM_FAULT_2: Fault input 2
 - HRTIM_FAULT_3: Fault input 3
 - HRTIM_FAULT_4: Fault input 4
 - HRTIM_FAULT_5: Fault input 5
- **Enable:** Fault(s) enabling This parameter can be one of the following values:
 - HRTIM_FAULTMODECTL_ENABLED: Fault(s) enabled
 - HRTIM_FAULTMODECTL_DISABLED: Fault(s) disabled

Return values

- **None:**

HAL_HRTIM_ADCTriggerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_ADCTriggerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t ADCTrigger, HRTIM_ADCTriggerCfgTypeDef * pADCTriggerCfg)

Function description

Configures both the ADC trigger register update source and the ADC trigger source.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **ADCTrigger:** ADC trigger to configure This parameter can be one of the following values:
 - HRTIM_ADCTRIGGER_1: ADC trigger 1
 - HRTIM_ADCTRIGGER_2: ADC trigger 2
 - HRTIM_ADCTRIGGER_3: ADC trigger 3
 - HRTIM_ADCTRIGGER_4: ADC trigger 4
- **pADCTriggerCfg:** pointer to the ADC trigger configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_WaveformTimerConfig

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformTimerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_TimerCfgTypeDef * pTimerCfg)

Function description

Configures the general behavior of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:

- HRTIM_TIMERINDEX_MASTER for master timer
- HRTIM_TIMERINDEX_TIMER_A for timer A
- HRTIM_TIMERINDEX_TIMER_B for timer B
- HRTIM_TIMERINDEX_TIMER_C for timer C
- HRTIM_TIMERINDEX_TIMER_D for timer D
- HRTIM_TIMERINDEX_TIMER_E for timer E
- **pTimerCfg:** pointer to the timer configuration structure

Return values

- **HAL:** status

Notes

- When the timer operates in waveform mode, all the features supported by the HRTIM are available without any limitation.
- This function must be called before starting the timer

HAL_HRTIM_WaveformCompareConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_WaveformCompareConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CompareUnit, HRTIM_CompareTypeDef * pCompareCfg)
```

Function description

Configures the compare unit of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CompareUnit:** Compare unit to configure This parameter can be one of the following values:
 - HRTIM_COMPAREUNIT_1: Compare unit 1
 - HRTIM_COMPAREUNIT_2: Compare unit 2
 - HRTIM_COMPAREUNIT_3: Compare unit 3
 - HRTIM_COMPAREUNIT_4: Compare unit 4
- **pCompareCfg:** pointer to the compare unit configuration structure

Return values

- **HAL:** status

Notes

- When auto delayed mode is required for compare unit 2 or compare unit 4, application has to configure separately the capture unit. Capture unit to configure in that case depends on the compare unit auto delayed mode is applied to (see below): Auto delayed on output compare 2: capture unit 1 must be configured Auto delayed on output compare 4: capture unit 2 must be configured
- This function must be called before starting the timer

HAL_HRTIM_WaveformCaptureConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_WaveformCaptureConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t CaptureUnit, HRTIM_CaptureCfgTypeDef * pCaptureCfg)
```

Function description

Configures the capture unit of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureUnit:** Capture unit to configure This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2
- **pCaptureCfg:** pointer to the compare unit configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_WaveformOutputConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_WaveformOutputConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, uint32_t Output, HRTIM_OutputCfgTypeDef * pOutputCfg)
```

Function description

Configures the output of a timer operating in waveform mode.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2

- HRTIM_OUTPUT_TD1: Timer D - Output 1
- HRTIM_OUTPUT_TD2: Timer D - Output 2
- HRTIM_OUTPUT_TE1: Timer E - Output 1
- HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pOutputCfg:** pointer to the timer output configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before configuring the timer and after configuring the deadtime insertion feature (if required).

HAL_HRTIM_WaveformSetOutputLevel

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformSetOutputLevel (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output, uint32_t OutputLevel)

Function description

Forces the timer output to its active or inactive state.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2
- **OutputLevel:** indicates whether the output is forced to its active or inactive level This parameter can be one of the following values:
 - HRTIM_OUTPUTLEVEL_ACTIVE: output is forced to its active level
 - HRTIM_OUTPUTLEVEL_INACTIVE: output is forced to its inactive level

Return values

- **HAL:** status

Notes

- The 'software set/reset trigger' bit in the output set/reset registers is automatically reset by hardware

HAL_HRTIM_TimerEventFilteringConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_TimerEventFilteringConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Event, HRTIM_TimerEventFilteringCfgTypeDef * pTimerEventFilteringCfg)
```

Function description

Configures the event filtering capabilities of a timer (blanking, windowing)

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Event:** external event for which timer event filtering must be configured This parameter can be one of the following values:
 - HRTIM_EVENT_NONE: Reset timer event filtering configuration
 - HRTIM_EVENT_1: External event 1
 - HRTIM_EVENT_2: External event 2
 - HRTIM_EVENT_3: External event 3
 - HRTIM_EVENT_4: External event 4
 - HRTIM_EVENT_5: External event 5
 - HRTIM_EVENT_6: External event 6
 - HRTIM_EVENT_7: External event 7
 - HRTIM_EVENT_8: External event 8
 - HRTIM_EVENT_9: External event 9
 - HRTIM_EVENT_10: External event 10
- **pTimerEventFilteringCfg:** pointer to the timer event filtering configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_DeadTimeConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_DeadTimeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_DeadTimeCfgTypeDef * pDeadTimeCfg)
```

Function description

Configures the deadtime insertion feature for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B

- HRTIM_TIMERINDEX_TIMER_C for timer C
- HRTIM_TIMERINDEX_TIMER_D for timer D
- HRTIM_TIMERINDEX_TIMER_E for timer E
- **pDeadTimeCfg:** pointer to the deadtime insertion configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_ChopperModeConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_ChopperModeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t  
TimerIdx, HRTIM_ChopperModeCfgTypeDef * pChopperModeCfg)
```

Function description

Configures the chopper mode feature for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **pChopperModeCfg:** pointer to the chopper mode configuration structure

Return values

- **HAL:** status

Notes

- This function must be called before configuring the timer output(s)

HAL_HRTIM_BurstDMAConfig

Function name

```
HAL_StatusTypeDef HAL_HRTIM_BurstDMAConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,  
uint32_t RegistersToUpdate)
```

Function description

Configures the burst DMA controller for a timer.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

- **RegistersToUpdate:** registers to be written by DMA This parameter can be any combination of the following values:
 - HRTIM_BURSTDMA_CR: HRTIM_MCR or HRTIM_TIMxCR
 - HRTIM_BURSTDMA_ICR: HRTIM_MICR or HRTIM_TIMxICR
 - HRTIM_BURSTDMA_DIER: HRTIM_MDIER or HRTIM_TIMxDIER
 - HRTIM_BURSTDMA_CNT: HRTIM_MCNT or HRTIM_TIMxCNT
 - HRTIM_BURSTDMA_PER: HRTIM_MP PER or HRTIM_TIMxPER
 - HRTIM_BURSTDMA REP: HRTIM_MREP or HRTIM_TIMxREP
 - HRTIM_BURSTDMA_CMP1: HRTIM_MCMP1 or HRTIM_TIMxCMP1
 - HRTIM_BURSTDMA_CMP2: HRTIM_MCMP2 or HRTIM_TIMxCMP2
 - HRTIM_BURSTDMA_CMP3: HRTIM_MCMP3 or HRTIM_TIMxCMP3
 - HRTIM_BURSTDMA_CMP4: HRTIM_MCMP4 or HRTIM_TIMxCMP4
 - HRTIM_BURSTDMA_DTR: HRTIM_TIMxDTR
 - HRTIM_BURSTDMA_SET1R: HRTIM_TIMxSET1R
 - HRTIM_BURSTDMA_RST1R: HRTIM_TIMxRST1R
 - HRTIM_BURSTDMA_SET2R: HRTIM_TIMxSET2R
 - HRTIM_BURSTDMA_RST2R: HRTIM_TIMxRST2R
 - HRTIM_BURSTDMA_EEFR1: HRTIM_TIMxEEFR1
 - HRTIM_BURSTDMA_EEFR2: HRTIM_TIMxEEFR2
 - HRTIM_BURSTDMA_RSTR: HRTIM_TIMxRSTR
 - HRTIM_BURSTDMA_CHPR: HRTIM_TIMxCHPR
 - HRTIM_BURSTDMA_OUTR: HRTIM_TIMxOUTR
 - HRTIM_BURSTDMA_FLTR: HRTIM_TIMxFLTR

Return values

- **HAL:** status

Notes

- This function must be called before starting the timer

HAL_HRTIM_WaveformCounterStart

Function name

```
HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStart (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)
```

Function description

Starts the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
 - HRTIM_TIMERID_MASTER
 - HRTIM_TIMERID_TIMER_A
 - HRTIM_TIMERID_TIMER_B
 - HRTIM_TIMERID_TIMER_C
 - HRTIM_TIMERID_TIMER_D
 - HRTIM_TIMERID_TIMER_E

Return values

- **HAL:** status

HAL_HRTIM_WaveformCounterStop

Function name

```
HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStop (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)
```

Function description

Stops the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
 - HRTIM_TIMER_MASTER
 - HRTIM_TIMER_A
 - HRTIM_TIMER_B
 - HRTIM_TIMER_C
 - HRTIM_TIMER_D
 - HRTIM_TIMER_E

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled

HAL_HRTIM_WaveformCounterStart_IT

Function name

```
HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)
```

Function description

Starts the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
 - HRTIM_TIMERID_MASTER
 - HRTIM_TIMERID_A
 - HRTIM_TIMERID_B
 - HRTIM_TIMERID_C
 - HRTIM_TIMERID_D
 - HRTIM_TIMERID_E

Return values

- **HAL:** status

Notes

- HRTIM interrupts (e.g. faults interrupts) and interrupts related to the timers to start are enabled within this function. Interrupts to enable are selected through HAL_HRTIM_WaveformTimerConfig function.

HAL_HRTIM_WaveformCounterStop_IT

Function name

```
HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)
```

Function description

Stops the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
 - HRTIM_TIMER_MASTER
 - HRTIM_TIMER_A
 - HRTIM_TIMER_B
 - HRTIM_TIMER_C
 - HRTIM_TIMER_D
 - HRTIM_TIMER_E

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled
- All enabled timer related interrupts are disabled.

HAL_HRTIM_WaveformCounterStart_DMA

Function name

```
HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)
```

Function description

Starts the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to start This parameter can be any combination of the following values:
HRTIM_TIMER_MASTER
 - HRTIM_TIMER_A
 - HRTIM_TIMER_B
 - HRTIM_TIMER_C
 - HRTIM_TIMER_D
 - HRTIM_TIMER_E

Return values

- **HAL:** status

Notes

- This function enables the dma request(s) mentionned in the timer configuration data structure for every timers to start.

- The source memory address, the destination memory address and the size of each DMA transfer are specified at timer configuration time (see HAL_HRTIM_WaveformTimerConfig)

HAL_HRTIM_WaveformCounterStop_DMA

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Stops the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values:
 - HRTIM_TIMER_MASTER
 - HRTIM_TIMER_A
 - HRTIM_TIMER_B
 - HRTIM_TIMER_C
 - HRTIM_TIMER_D
 - HRTIM_TIMER_E

Return values

- **HAL:** status

Notes

- The counter of a timer is stopped only if all timer outputs are disabled
- All enabled timer related DMA requests are disabled.

HAL_HRTIM_WaveformOutputStart

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformOutputStart (HRTIM_HandleTypeDef * hhrtim, uint32_t OutputsToStart)

Function description

Enables the generation of the waveform signal on the designated output(s) Outputs can be combined (ORed) to allow for simultaneous output enabling.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **OutputsToStart:** Timer output(s) to enable This parameter can be any combination of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_WaveformOutputStop

Function name

HAL_StatusTypeDef HAL_HRTIM_WaveformOutputStop (HRTIM_HandleTypeDef * hhrtim, uint32_t OutputsToStop)

Function description

Disables the generation of the waveform signal on the designated output(s) Outputs can be combined (ORed) to allow for simultaneous output disabling.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **OutputsToStop:** Timer output(s) to disable This parameter can be any combination of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **HAL:** status

HAL_HRTIM_BurstModeCtl

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeCtl (HRTIM_HandleTypeDef * hhrtim, uint32_t Enable)

Function description

Enables or disables the HRTIM burst mode controller.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Enable:** Burst mode controller enabling This parameter can be one of the following values:
 - HRTIM_BURSTMODECTL_ENABLED: Burst mode enabled
 - HRTIM_BURSTMODECTL_DISABLED: Burst mode disabled

Return values

- **HAL:** status

Notes

- This function must be called after starting the timer(s)

HAL_HRTIM_BurstModeSoftwareTrigger

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstModeSoftwareTrigger (HRTIM_HandleTypeDef * hhrtim)

Function description

Triggers the burst mode operation.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** status

HAL_HRTIM_SoftwareCapture

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareCapture (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)

Function description

Triggers a software capture on the designed capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureUnit:** Capture unit to trig This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **HAL:** status

Notes

- The 'software capture' bit in the capure configuration register is automatically reset by hardware

HAL_HRTIM_SoftwareUpdate

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareUpdate (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Triggers the update of the registers of one or several timers.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software register update This parameter can be any combination of the following values:
 - HRTIM_TIMERUPDATE_MASTER
 - HRTIM_TIMERUPDATE_A
 - HRTIM_TIMERUPDATE_B
 - HRTIM_TIMERUPDATE_C
 - HRTIM_TIMERUPDATE_D
 - HRTIM_TIMERUPDATE_E

Return values

- **HAL:** status

Notes

- The 'software update' bits in the HRTIM control register 2 register are automatically reset by hardware

HAL_HRTIM_SoftwareReset

Function name

HAL_StatusTypeDef HAL_HRTIM_SoftwareReset (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Triggers the reset of one or several timers.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** timers concerned with the software counter reset This parameter can be any combination of the following values:
 - HRTIM_TIMERRESET_MASTER
 - HRTIM_TIMERRESET_TIMER_A
 - HRTIM_TIMERRESET_TIMER_B
 - HRTIM_TIMERRESET_TIMER_C
 - HRTIM_TIMERRESET_TIMER_D
 - HRTIM_TIMERRESET_TIMER_E

Return values

- **HAL:** status

Notes

- The 'software reset' bits in the HRTIM control register 2 are automatically reset by hardware

HAL_HRTIM_BurstDMATransfer

Function name

HAL_StatusTypeDef HAL_HRTIM_BurstDMATransfer (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t BurstBufferAddress, uint32_t BurstBufferLength)

Function description

Starts a burst DMA operation to update HRTIM control registers content.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **BurstBufferAddress:** address of the buffer the HRTIM control registers content will be updated from.
- **BurstBufferLength:** size (in WORDS) of the burst buffer.

Return values

- **HAL:** status

Notes

- The TimerIdx parameter determines the dma channel to be used by the DMA burst controller (see below)
HRTIM_TIMERINDEX_MASTER: DMA channel 2 is used by the DMA burst controller
HRTIM_TIMERINDEX_TIMER_A: DMA channel 3 is used by the DMA burst controller
HRTIM_TIMERINDEX_TIMER_B: DMA channel 4 is used by the DMA burst controller
HRTIM_TIMERINDEX_TIMER_C: DMA channel 5 is used by the DMA burst controller
HRTIM_TIMERINDEX_TIMER_D: DMA channel 6 is used by the DMA burst controller
HRTIM_TIMERINDEX_TIMER_E: DMA channel 7 is used by the DMA burst controller

HAL_HRTIM_UpdateEnable

Function name

HAL_StatusTypeDef HAL_HRTIM_UpdateEnable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Enables the transfer from preload to active registers for one or several timing units (including master timer).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer(s) concerned by the register preload enabling command This parameter can be any combination of the following values:
 - HRTIM_TIMERUPDATE_MASTER
 - HRTIM_TIMERUPDATE_A
 - HRTIM_TIMERUPDATE_B
 - HRTIM_TIMERUPDATE_C
 - HRTIM_TIMERUPDATE_D
 - HRTIM_TIMERUPDATE_E

Return values

- **HAL:** status

HAL_HRTIM_UpdateDisable

Function name

HAL_StatusTypeDef HAL_HRTIM_UpdateDisable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)

Function description

Disables the transfer from preload to active registers for one or several timing units (including master timer).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **Timers:** Timer(s) concerned by the register preload disabling command This parameter can be any combination of the following values:
 - HRTIM_TIMERUPDATE_MASTER
 - HRTIM_TIMERUPDATE_A
 - HRTIM_TIMERUPDATE_B
 - HRTIM_TIMERUPDATE_C
 - HRTIM_TIMERUPDATE_D
 - HRTIM_TIMERUPDATE_E

Return values

- **HAL:** status

HAL_HRTIM_GetState

Function name

```
HAL_HRTIM_StateTypeDef HAL_HRTIM_GetState (HRTIM_HandleTypeDef * hhrtim)
```

Function description

return the HRTIM HAL state

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **HAL:** state

HAL_HRTIM_GetCapturedValue

Function name

```
uint32_t HAL_HRTIM_GetCapturedValue (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)
```

Function description

Returns actual value of the capture register of the designated capture unit.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **CaptureUnit:** Capture unit to trig This parameter can be one of the following values:
 - HRTIM_CAPTUREUNIT_1: Capture unit 1
 - HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values

- **Captured:** value

HAL_HRTIM_WaveformGetOutputLevel

Function name

```
uint32_t HAL_HRTIM_WaveformGetOutputLevel (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)
```

Function description

Returns actual level (active or inactive) of the designated output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D

- HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **Output:** level

Notes

- Returned output level is taken before the output stage (chopper, polarity).

HAL_HRTIM_WaveformGetOutputState

Function name

```
uint32_t HAL_HRTIM_WaveformGetOutputState (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,  
uint32_t Output)
```

Function description

Returns actual state (RUN, IDLE, FAULT) of the designated output.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **Output:** state

HAL_HRTIM_GetDelayedProtectionStatus

Function name

```
uint32_t HAL_HRTIM_GetDelayedProtectionStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx,  
uint32_t Output)
```

Function description

Returns the level (active or inactive) of the designated output when the delayed protection was triggered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values:
 - HRTIM_OUTPUT_TA1: Timer A - Output 1
 - HRTIM_OUTPUT_TA2: Timer A - Output 2
 - HRTIM_OUTPUT_TB1: Timer B - Output 1
 - HRTIM_OUTPUT_TB2: Timer B - Output 2
 - HRTIM_OUTPUT_TC1: Timer C - Output 1
 - HRTIM_OUTPUT_TC2: Timer C - Output 2
 - HRTIM_OUTPUT_TD1: Timer D - Output 1
 - HRTIM_OUTPUT_TD2: Timer D - Output 2
 - HRTIM_OUTPUT_TE1: Timer E - Output 1
 - HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values

- **Delayed:** protection status

HAL_HRTIM_GetBurstStatus

Function name

```
uint32_t HAL_HRTIM_GetBurstStatus (HRTIM_HandleTypeDef * hhrtim)
```

Function description

Returns the actual status (active or inactive) of the burst mode controller.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **Burst:** mode controller status

HAL_HRTIM_GetCurrentPushPullStatus

Function name

```
uint32_t HAL_HRTIM_GetCurrentPushPullStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)
```

Function description

Indicates on which output the signal is currently active (when the push pull mode is enabled).

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **Burst:** mode controller status

HAL_HRTIM_GetIdlePushPullStatus

Function name

uint32_t HAL_HRTIM_GetIdlePushPullStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Indicates on which output the signal was applied, in push-pull mode, balanced fault mode or delayed idle mode, when the protection was triggered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **Idle:** Push Pull Status

HAL_HRTIM_IRQHandler

Function name

void HAL_HRTIM_IRQHandler (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

This function handles HRTIM interrupt request.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be any value of HRTIM Timer Index

Return values

- **None:**

HAL_HRTIM_Fault1Callback

Function name

void HAL_HRTIM_Fault1Callback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a fault 1 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle *

Return values

- **None:**
- **None:**

HAL_HRTIM_Fault2Callback**Function name**

```
void HAL_HRTIM_Fault2Callback (HRTIM_HandleTypeDef * hhrtim)
```

Function description

Callback function invoked when a fault 2 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault3Callback**Function name**

```
void HAL_HRTIM_Fault3Callback (HRTIM_HandleTypeDef * hhrtim)
```

Function description

Callback function invoked when a fault 3 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault4Callback**Function name**

```
void HAL_HRTIM_Fault4Callback (HRTIM_HandleTypeDef * hhrtim)
```

Function description

Callback function invoked when a fault 4 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_Fault5Callback**Function name**

```
void HAL_HRTIM_Fault5Callback (HRTIM_HandleTypeDef * hhrtim)
```

Function description

Callback function invoked when a fault 5 interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_SystemFaultCallback

Function name

void HAL_HRTIM_SystemFaultCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a system fault interrupt occurred.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_BurstModePeriodCallback

Function name

void HAL_HRTIM_BurstModePeriodCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when the end of the burst mode period is reached.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_SynchronizationEventCallback

Function name

void HAL_HRTIM_SynchronizationEventCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a synchronization input event is received.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

HAL_HRTIM_RegistersUpdateCallback

Function name

void HAL_HRTIM_RegistersUpdateCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when timer registers are updated.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_RepetitionEventCallback

Function name

void HAL_HRTIM_RepetitionEventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when timer repetition period has elapsed.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Compare1EventCallback

Function name

void HAL_HRTIM_Compare1EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 1 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Compare2EventCallback

Function name

void HAL_HRTIM_Compare2EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 2 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Compare3EventCallback

Function name

void HAL_HRTIM_Compare3EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 3 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Compare4EventCallback

Function name

void HAL_HRTIM_Compare4EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer counter matches the value programmed in the compare 4 register.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Capture1EventCallback

Function name

void HAL_HRTIM_Capture1EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x capture 1 event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Capture2EventCallback

Function name

void HAL_HRTIM_Capture2EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x capture 2 event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_DelayedProtectionCallback

Function name

```
void HAL_HRTIM_DelayedProtectionCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)
```

Function description

Callback function invoked when the delayed idle or balanced idle mode is entered.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_CounterResetCallback

Function name

```
void HAL_HRTIM_CounterResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)
```

Function description

Callback function invoked when the timer x counter reset/roll-over event occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Output1SetCallback

Function name

```
void HAL_HRTIM_Output1SetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)
```

Function description

Callback function invoked when the timer x output 1 is set.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C

- HRTIM_TIMERINDEX_TIMER_D for timer D
- HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Output1ResetCallback

Function name

void HAL_HRTIM_Output1ResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 1 is reset.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Output2SetCallback

Function name

void HAL_HRTIM_Output2SetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 2 is set.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_Output2ResetCallback

Function name

void HAL_HRTIM_Output2ResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when the timer x output 2 is reset.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_BurstDMATransferCallback

Function name

void HAL_HRTIM_BurstDMATransferCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function description

Callback function invoked when a DMA burst transfer is completed.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values:
 - HRTIM_TIMERINDEX_MASTER for master timer
 - HRTIM_TIMERINDEX_TIMER_A for timer A
 - HRTIM_TIMERINDEX_TIMER_B for timer B
 - HRTIM_TIMERINDEX_TIMER_C for timer C
 - HRTIM_TIMERINDEX_TIMER_D for timer D
 - HRTIM_TIMERINDEX_TIMER_E for timer E

Return values

- **None:**

HAL_HRTIM_ErrorCallback

Function name

void HAL_HRTIM_ErrorCallback (HRTIM_HandleTypeDef * hhrtim)

Function description

Callback function invoked when a DMA error occurs.

Parameters

- **hhrtim:** pointer to HAL HRTIM handle

Return values

- **None:**

31.3 HRTIM Firmware driver defines

31.3.1 HRTIM

HRTIM ADC Trigger

HRTIM_ADCTRIGGER_1

ADC trigger 1 identifier.

HRTIM_ADCTRIGGER_2

ADC trigger 2 identifier.

HRTIM_ADCTRIGGER_3

ADC trigger 3 identifier.

HRTIM_ADCTRIGGER_4

ADC trigger 4 identifier.

HRTIM ADC Trigger Event**HRTIM_ADCTRIGGEREVENT13_NONE**

No ADC trigger event.

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP1

ADC Trigger on master compare 1.

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP2

ADC Trigger on master compare 2.

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP3

ADC Trigger on master compare 3.

HRTIM_ADCTRIGGEREVENT13_MASTER_CMP4

ADC Trigger on master compare 4.

HRTIM_ADCTRIGGEREVENT13_MASTER_PERIOD

ADC Trigger on master period.

HRTIM_ADCTRIGGEREVENT13_EVENT_1

ADC Trigger on external event 1.

HRTIM_ADCTRIGGEREVENT13_EVENT_2

ADC Trigger on external event 2.

HRTIM_ADCTRIGGEREVENT13_EVENT_3

ADC Trigger on external event 3.

HRTIM_ADCTRIGGEREVENT13_EVENT_4

ADC Trigger on external event 4.

HRTIM_ADCTRIGGEREVENT13_EVENT_5

ADC Trigger on external event 5.

HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP2

ADC Trigger on Timer A compare 2.

HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP3

ADC Trigger on Timer A compare 3.

HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP4

ADC Trigger on Timer A compare 4.

HRTIM_ADCTRIGGEREVENT13_TIMERA_PERIOD

ADC Trigger on Timer A period.

HRTIM_ADCTRIGGEREVENT13_TIMERA_RESET

ADC Trigger on Timer A reset.

HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP2

ADC Trigger on Timer B compare 2.

HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP3

ADC Trigger on Timer B compare 3.

HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP4

ADC Trigger on Timer B compare 4.

HRTIM_ADCTRIGGEREVENT13_TIMERB_PERIOD

ADC Trigger on Timer B period.

HRTIM_ADCTRIGGEREVENT13_TIMERB_RESET

ADC Trigger on Timer B reset.

HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP2

ADC Trigger on Timer C compare 2.

HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP3

ADC Trigger on Timer C compare 3.

HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP4

ADC Trigger on Timer C compare 4.

HRTIM_ADCTRIGGEREVENT13_TIMERC_PERIOD

ADC Trigger on Timer C period.

HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP2

ADC Trigger on Timer D compare 2.

HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP3

ADC Trigger on Timer D compare 3.

HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP4

ADC Trigger on Timer D compare 4.

HRTIM_ADCTRIGGEREVENT13_TIMERD_PERIOD

ADC Trigger on Timer D period.

HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP2

ADC Trigger on Timer E compare 2.

HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP3

ADC Trigger on Timer E compare 3.

HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP4

ADC Trigger on Timer E compare 4.

HRTIM_ADCTRIGGEREVENT13_TIMERE_PERIOD

ADC Trigger on Timer E period.

HRTIM_ADCTRIGGEREVENT24_NONE

No ADC trigger event.

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP1

ADC Trigger on master compare 1.

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP2

ADC Trigger on master compare 2.

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP3

ADC Trigger on master compare 3.

HRTIM_ADCTRIGGEREVENT24_MASTER_CMP4

ADC Trigger on master compare 4.

HRTIM_ADCTRIGGEREVENT24_MASTER_PERIOD

ADC Trigger on master period.

HRTIM_ADCTRIGGEREVENT24_EVENT_6

ADC Trigger on external event 6.

HRTIM_ADCTRIGGEREVENT24_EVENT_7

ADC Trigger on external event 7.

HRTIM_ADCTRIGGEREVENT24_EVENT_8

ADC Trigger on external event 8.

HRTIM_ADCTRIGGEREVENT24_EVENT_9

ADC Trigger on external event 9.

HRTIM_ADCTRIGGEREVENT24_EVENT_10

ADC Trigger on external event 10.

HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP2

ADC Trigger on Timer A compare 2.

HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP3

ADC Trigger on Timer A compare 3.

HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP4

ADC Trigger on Timer A compare 4.

HRTIM_ADCTRIGGEREVENT24_TIMERA_PERIOD

ADC Trigger on Timer A period.

HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP2

ADC Trigger on Timer B compare 2.

HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP3

ADC Trigger on Timer B compare 3.

HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP4

ADC Trigger on Timer B compare 4.

HRTIM_ADCTRIGGEREVENT24_TIMERB_PERIOD

ADC Trigger on Timer B period.

HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP2

ADC Trigger on Timer C compare 2.

HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP3

ADC Trigger on Timer C compare 3.

HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP4

ADC Trigger on Timer C compare 4.

HRTIM_ADCTRIGGEREVENT24_TIMERC_PERIOD

ADC Trigger on Timer C period.

HRTIM_ADCTRIGGEREVENT24_TIMERC_RESET

ADC Trigger on Timer C reset.

HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP2

ADC Trigger on Timer D compare 2.

HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP3

ADC Trigger on Timer D compare 3.

HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP4

ADC Trigger on Timer D compare 4.

HRTIM_ADCTRIGGEREVENT24_TIMERD_PERIOD

ADC Trigger on Timer D period.

HRTIM_ADCTRIGGEREVENT24_TIMERD_RESET

ADC Trigger on Timer D reset.

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP2

ADC Trigger on Timer E compare 2 .

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP3

ADC Trigger on Timer E compare 3.

HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP4

ADC Trigger on Timer E compare 4.

HRTIM_ADCTRIGGEREVENT24_TIMERE_RESET

ADC Trigger on Timer E reset.

HRTIM ADC Trigger Update Source**HRTIM_ADCTRIGGERUPDATE_MASTER**

Master timer.

HRTIM_ADCTRIGGERUPDATE_TIMER_A

Timer A.

HRTIM_ADCTRIGGERUPDATE_TIMER_B

Timer B.

HRTIM_ADCTRIGGERUPDATE_TIMER_C

Timer C.

HRTIM_ADCTRIGGERUPDATE_TIMER_D

Timer D.

HRTIM_ADCTRIGGERUPDATE_TIMER_E

Timer E.

HRTIM Burst DMA Registers Update**HRTIM_BURSTDMA_NONE**

No register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_CR

MCR or TIMxCR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_ICR

MICR or TIMxICR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_DIER

MDIER or TIMxDIER register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_CNT

MCNTR or CNTxCR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_PER

MPER or PERxR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA REP

MREPR or REPxR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_CMP1

MCMP1R or CMP1xR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_CMP2

MCMP2R or CMP2xR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_CMP3

MCMP3R or CMP3xR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_CMP4

MCMP4R or CMP4xR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_DTR

TDxR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_SET1R

SET1R register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_RST1R

RST1R register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_SET2R

SET2R register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_RST2R

RST1R register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_EEFR1

EEFxR1 register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_EEFR2

EEFxR2 register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_RSTR

RSTxR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_CHPR

CHPxR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_OUTR

OUTxR register is updated by Burst DMA accesses.

HRTIM_BURSTDMA_FLTR

FLTxD register is updated by Burst DMA accesses.

HRTIM Burst Mode Clock Source**HRTIM_BURSTMODECLOCKSOURCE_MASTER**

Master timer counter reset/roll-over is used as clock source for the burst mode counter.

HRTIM_BURSTMODECLOCKSOURCE_TIMER_A

Timer A counter reset/roll-over is used as clock source for the burst mode counter.

HRTIM_BURSTMODECLOCKSOURCE_TIMER_B

Timer B counter reset/roll-over is used as clock source for the burst mode counter.

HRTIM_BURSTMODECLOCKSOURCE_TIMER_C

Timer C counter reset/roll-over is used as clock source for the burst mode counter.

HRTIM_BURSTMODECLOCKSOURCE_TIMER_D

Timer D counter reset/roll-over is used as clock source for the burst mode counter.

HRTIM_BURSTMODECLOCKSOURCE_TIMER_E

Timer E counter reset/roll-over is used as clock source for the burst mode counter.

HRTIM_BURSTMODECLOCKSOURCE_TIM16_OC

On-chip Event 1 (BMClk[1]), acting as a burst mode counter clock.

HRTIM_BURSTMODECLOCKSOURCE_TIM17_OC

On-chip Event 2 (BMClk[2]), acting as a burst mode counter clock.

HRTIM_BURSTMODECLOCKSOURCE_TIM7_TRGO

On-chip Event 3 (BMClk[3]), acting as a burst mode counter clock.

HRTIM_BURSTMODECLOCKSOURCE_FHRTIM

Prescaled fHRTIM clock is used as clock source for the burst mode counter.

HRTIM Burst Mode Control**HRTIM_BURSTMODECTL_DISABLED**

Burst mode disabled.

HRTIM_BURSTMODECTL_ENABLED

Burst mode enabled.

HRTIM Burst Mode Operating Mode**HRTIM_BURSTMODE_SINGLESHOT**

Burst mode operates in single shot mode.

HRTIM_BURSTMODE_CONTINUOUS

Burst mode operates in continuous mode.

HRTIM Burst Mode Prescaler**HRTIM_BURSTMODEPRESCALER_DIV1**

fBRST = fHRTIM.

HRTIM_BURSTMODEPRESCALER_DIV2

fBRST = fHRTIM/2.

HRTIM_BURSTMODEPRESCALER_DIV4

fBRST = fHRTIM/4.

HRTIM_BURSTMODEPRESCALER_DIV8

fBRST = fHRTIM/8.

HRTIM_BURSTMODEPRESCALER_DIV16

fBRST = fHRTIM/16.

HRTIM_BURSTMODEPRESCALER_DIV32

fBRST = fHRTIM/32.

HRTIM_BURSTMODEPRESCALER_DIV64

fBRST = fHRTIM/64.

HRTIM_BURSTMODEPRESCALER_DIV128

fBRST = fHRTIM/128.

HRTIM_BURSTMODEPRESCALER_DIV256

fBRST = fHRTIM/256.

HRTIM_BURSTMODEPRESCALER_DIV512

fBRST = fHRTIM/512.

HRTIM_BURSTMODEPRESCALER_DIV1024

fBRST = fHRTIM/1024.

HRTIM_BURSTMODEPRESCALER_DIV2048

fBRST = fHRTIM/2048.

HRTIM_BURSTMODEPRESCALER_DIV4096

fBRST = fHRTIM/4096.

HRTIM_BURSTMODEPRESCALER_DIV8192

fBRST = fHRTIM/8192.

HRTIM_BURSTMODEPRESCALER_DIV16384

fBRST = fHRTIM/16384.

HRTIM_BURSTMODEPRESCALER_DIV32768

fBRST = fHRTIM/32768.

HRTIM Burst Mode Register Preload Enable**`HRTIM_BURSTMODEPRELOAD_DISABLED`**

Preload disabled: the write access is directly done into active registers.

`HRTIM_BURSTMODEPRELOAD_ENABLED`

Preload enabled: the write access is done into preload registers.

HRTIM Burst Mode Status**`HRTIM_BURSTMODESTATUS_NORMAL`**

Normal operation.

`HRTIM_BURSTMODESTATUS_ONGOING`

Burst operation on-going.

HRTIM Burst Mode Trigger**`HRTIM_BURSTMODEtrigger_NONE`**

No trigger.

`HRTIM_BURSTMODEtrigger_MASTER_RESET`

Master reset.

`HRTIM_BURSTMODEtrigger_MASTER_REPEATITION`

Master repetition.

`HRTIM_BURSTMODEtrigger_MASTER_CMP1`

Master compare 1.

`HRTIM_BURSTMODEtrigger_MASTER_CMP2`

Master compare 2.

`HRTIM_BURSTMODEtrigger_MASTER_CMP3`

Master compare 3.

`HRTIM_BURSTMODEtrigger_MASTER_CMP4`

Master compare 4.

`HRTIM_BURSTMODEtrigger_TIMERA_RESET`

Timer A reset.

`HRTIM_BURSTMODEtrigger_TIMERA_REPEATITION`

Timer A repetition.

`HRTIM_BURSTMODEtrigger_TIMERA_CMP1`

Timer A compare 1.

`HRTIM_BURSTMODEtrigger_TIMERA_CMP2`

Timer A compare 2.

`HRTIM_BURSTMODEtrigger_TIMERB_RESET`

Timer B reset.

`HRTIM_BURSTMODEtrigger_TIMERB_REPEATITION`

Timer B repetition.

HRTIM_BURSTMODETRIGGER_TIMERB_CMP1

Timer B compare 1.

HRTIM_BURSTMODETRIGGER_TIMERB_CMP2

Timer B compare 2.

HRTIM_BURSTMODETRIGGER_TIMERC_RESET

Timer C reset.

HRTIM_BURSTMODETRIGGER_TIMERC_REPETITION

Timer C repetition.

HRTIM_BURSTMODETRIGGER_TIMERC_CMP1

Timer C compare 1.

HRTIM_BURSTMODETRIGGER_TIMERC_CMP2

Timer C compare 2.

HRTIM_BURSTMODETRIGGER_TIMERD_RESET

Timer D reset.

HRTIM_BURSTMODETRIGGER_TIMERD_REPETITION

Timer D repetition.

HRTIM_BURSTMODETRIGGER_TIMERD_CMP1

Timer D compare 1.

HRTIM_BURSTMODETRIGGER_TIMERD_CMP2

Timer D compare 2.

HRTIM_BURSTMODETRIGGER_TIMERE_RESET

Timer E reset.

HRTIM_BURSTMODETRIGGER_TIMERE_REPETITION

Timer E repetition.

HRTIM_BURSTMODETRIGGER_TIMERE_CMP1

Timer E compare 1.

HRTIM_BURSTMODETRIGGER_TIMERE_CMP2

Timer E compare 2.

HRTIM_BURSTMODETRIGGER_TIMERA_EVENT7

Timer A period following External Event 7.

HRTIM_BURSTMODETRIGGER_TIMERD_EVENT8

Timer D period following External Event 8.

HRTIM_BURSTMODETRIGGER_EVENT_7

External Event 7 (timer A filters applied).

HRTIM_BURSTMODETRIGGER_EVENT_8

External Event 8 (timer D filters applied).

HRTIM_BURSTMODETRIGGER_EVENT_ONCHIP

On-chip Event

HRTIM Capture Unit**HRTIM_CAPTUREUNIT_1**

Capture unit 1 identifier.

HRTIM_CAPTUREUNIT_2

Capture unit 2 identifier.

HRTIM Capture Unit Trigger**HRTIM_CAPTURETRIGGER_NONE**

Capture trigger is disabled.

HRTIM_CAPTURETRIGGER_UPDATE

The update event triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_1

The External event 1 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_2

The External event 2 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_3

The External event 3 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_4

The External event 4 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_5

The External event 5 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_6

The External event 6 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_7

The External event 7 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_8

The External event 8 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_9

The External event 9 triggers the Capture.

HRTIM_CAPTURETRIGGER_EEV_10

The External event 10 triggers the Capture.

HRTIM_CAPTURETRIGGER_TA1_SET

Capture is triggered by TA1 output inactive to active transition.

HRTIM_CAPTURETRIGGER_TA1_RESET

Capture is triggered by TA1 output active to inactive transition.

HRTIM_CAPTURETRIGGER_TIMERA_CMP1

Timer A Compare 1 triggers Capture.

HRTIM_CAPTURETRIGGER_TIMERA_CMP2

Timer A Compare 2 triggers Capture.

HRTIM_CAPTURETRIGGER_TB1_SET

Capture is triggered by TB1 output inactive to active transition.

HRTIM_CAPTURETRIGGER_TB1_RESET

Capture is triggered by TB1 output active to inactive transition.

HRTIM_CAPTURETRIGGER_TIMERB_CMP1

Timer B Compare 1 triggers Capture.

HRTIM_CAPTURETRIGGER_TIMERB_CMP2

Timer B Compare 2 triggers Capture.

HRTIM_CAPTURETRIGGER_TC1_SET

Capture is triggered by TC1 output inactive to active transition.

HRTIM_CAPTURETRIGGER_TC1_RESET

Capture is triggered by TC1 output active to inactive transition.

HRTIM_CAPTURETRIGGER_TIMERC_CMP1

Timer C Compare 1 triggers Capture.

HRTIM_CAPTURETRIGGER_TIMERC_CMP2

Timer C Compare 2 triggers Capture.

HRTIM_CAPTURETRIGGER_TD1_SET

Capture is triggered by TD1 output inactive to active transition.

HRTIM_CAPTURETRIGGER_TD1_RESET

Capture is triggered by TD1 output active to inactive transition.

HRTIM_CAPTURETRIGGER_TIMERD_CMP1

Timer D Compare 1 triggers Capture.

HRTIM_CAPTURETRIGGER_TIMERD_CMP2

Timer D Compare 2 triggers Capture.

HRTIM_CAPTURETRIGGER_TE1_SET

Capture is triggered by TE1 output inactive to active transition.

HRTIM_CAPTURETRIGGER_TE1_RESET

Capture is triggered by TE1 output active to inactive transition.

HRTIM_CAPTURETRIGGER_TIMERE_CMP1

Timer E Compare 1 triggers Capture.

HRTIM_CAPTURETRIGGER_TIMERE_CMP2

Timer E Compare 2 triggers Capture.

HRTIM Chopper Duty Cycle**HRTIM_CHOPPER_DUTYCYCLE_0**

Only 1st pulse is present.

HRTIM_CHOPPER_DUTYCYCLE_125

Duty cycle of the carrier signal is 12.5 %.

HRTIM_CHOPPER_DUTYCYCLE_250

Duty cycle of the carrier signal is 25 %.

HRTIM_CHOPPER_DUTYCYCLE_375

Duty cycle of the carrier signal is 37.5 %.

HRTIM_CHOPPER_DUTYCYCLE_500

Duty cycle of the carrier signal is 50 %.

HRTIM_CHOPPER_DUTYCYCLE_625

Duty cycle of the carrier signal is 62.5 %.

HRTIM_CHOPPER_DUTYCYCLE_750

Duty cycle of the carrier signal is 75 %.

HRTIM_CHOPPER_DUTYCYCLE_875

Duty cycle of the carrier signal is 87.5 %.

HRTIM Chopper Frequency**HRTIM_CHOPPER_PRESCALERRATIO_DIV16**

$f_{CHPFRQ} = f_{HRTIM} / 16.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV32

$f_{CHPFRQ} = f_{HRTIM} / 32.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV48

$f_{CHPFRQ} = f_{HRTIM} / 48.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV64

$f_{CHPFRQ} = f_{HRTIM} / 64.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV80

$f_{CHPFRQ} = f_{HRTIM} / 80.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV96

$f_{CHPFRQ} = f_{HRTIM} / 96.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV112

$f_{CHPFRQ} = f_{HRTIM} / 112.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV128

$f_{CHPFRQ} = f_{HRTIM} / 128.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV144

$f_{CHPFRQ} = f_{HRTIM} / 144.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV160

$f_{CHPFRQ} = f_{HRTIM} / 160.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV176

$f_{CHPFRQ} = f_{HRTIM} / 176.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV192

$f_{CHPFRQ} = f_{HRTIM} / 192.$

HRTIM_CHOPPER_PRESCALERRATIO_DIV208

fCHPFRQ = fHRTIM / 208.

HRTIM_CHOPPER_PRESCALERRATIO_DIV224

fCHPFRQ = fHRTIM / 224.

HRTIM_CHOPPER_PRESCALERRATIO_DIV240

fCHPFRQ = fHRTIM / 240.

HRTIM_CHOPPER_PRESCALERRATIO_DIV256

fCHPFRQ = fHRTIM / 256.

HRTIM Chopper Start Pulse Width**HRTIM_CHOPPER_PULSEWIDTH_16**

tSTPW = tHRTIM x 16.

HRTIM_CHOPPER_PULSEWIDTH_32

tSTPW = tHRTIM x 32.

HRTIM_CHOPPER_PULSEWIDTH_48

tSTPW = tHRTIM x 48.

HRTIM_CHOPPER_PULSEWIDTH_64

tSTPW = tHRTIM x 64.

HRTIM_CHOPPER_PULSEWIDTH_80

tSTPW = tHRTIM x 80.

HRTIM_CHOPPER_PULSEWIDTH_96

tSTPW = tHRTIM x 96.

HRTIM_CHOPPER_PULSEWIDTH_112

tSTPW = tHRTIM x 112.

HRTIM_CHOPPER_PULSEWIDTH_128

tSTPW = tHRTIM x 128.

HRTIM_CHOPPER_PULSEWIDTH_144

tSTPW = tHRTIM x 144.

HRTIM_CHOPPER_PULSEWIDTH_160

tSTPW = tHRTIM x 160.

HRTIM_CHOPPER_PULSEWIDTH_176

tSTPW = tHRTIM x 176.

HRTIM_CHOPPER_PULSEWIDTH_192

tSTPW = tHRTIM x 192.

HRTIM_CHOPPER_PULSEWIDTH_208

tSTPW = tHRTIM x 208.

HRTIM_CHOPPER_PULSEWIDTH_224

tSTPW = tHRTIM x 224.

HRTIM_CHOPPER_PULSEWIDTH_240

tSTPW = tHRTIM x 240.

HRTIM_CHOPPER_PULSEWIDTH_256

tSTPW = tHRTIM x 256.

HRTIM Common Interrupt Enable**HRTIM_IT_NONE**

No interrupt enabled.

HRTIM_IT_FLT1

Fault 1 interrupt enable.

HRTIM_IT_FLT2

Fault 2 interrupt enable.

HRTIM_IT_FLT3

Fault 3 interrupt enable.

HRTIM_IT_FLT4

Fault 4 interrupt enable.

HRTIM_IT_FLT5

Fault 5 interrupt enable.

HRTIM_IT_SYSFLT

System Fault interrupt enable.

HRTIM_IT_BMPER

Burst mode period interrupt enable.

HRTIM Common Interrupt Flag**HRTIM_FLAG_FLT1**

Fault 1 interrupt flag.

HRTIM_FLAG_FLT2

Fault 2 interrupt flag.

HRTIM_FLAG_FLT3

Fault 3 interrupt flag.

HRTIM_FLAG_FLT4

Fault 4 interrupt flag.

HRTIM_FLAG_FLT5

Fault 5 interrupt flag.

HRTIM_FLAG_SYSFLT

System Fault interrupt flag.

HRTIM_FLAG_BMPER

Burst mode period interrupt flag.

HRTIM Compare Unit

HRTIM_COMPAREUNIT_1

Compare unit 1 identifier.

HRTIM_COMPAREUNIT_2

Compare unit 2 identifier.

HRTIM_COMPAREUNIT_3

Compare unit 3 identifier.

HRTIM_COMPAREUNIT_4

Compare unit 4 identifier.

HRTIM Compare Unit Auto Delayed Mode**HRTIM_AUTODELAYEDMODE_REGULAR**

standard compare mode.

HRTIM_AUTODELAYEDMODE_AUTODELAYED_NOTIMEOUT

Compare event generated only if a capture has occurred.

HRTIM_AUTODELAYEDMODE_AUTODELAYED_TIMEOUTCMP1

Compare event generated if a capture has occurred or after a Compare 1 match (timeout if capture event is missing).

HRTIM_AUTODELAYEDMODE_AUTODELAYED_TIMEOUTCMP3

Compare event generated if a capture has occurred or after a Compare 3 match (timeout if capture event is missing).

HRTIM Current Push Pull Status**HRTIM_PUSHULL_CURRENTSTATUS_OUTPUT1**

Signal applied on output 1 and output 2 forced inactive.

HRTIM_PUSHULL_CURRENTSTATUS_OUTPUT2

Signal applied on output 2 and output 1 forced inactive.

HRTIM DAC Synchronization**HRTIM_DACSYNC_NONE**

No DAC synchronization event generated.

HRTIM_DACSYNC_DACTRIGOUT_1

DAC synchronization event generated on DACTrigOut1 output upon timer update.

HRTIM_DACSYNC_DACTRIGOUT_2

DAC synchronization event generated on DACTrigOut2 output upon timer update.

HRTIM_DACSYNC_DACTRIGOUT_3

DAC update generated on DACTrigOut3 output upon timer update.

HRTIM Deadtime Falling Lock**HRTIM_TIMDEADTIME_FALLINGLOCK_WRITE**

Deadtime falling value and sign is writeable.

HRTIM_TIMDEADTIME_FALLINGLOCK_READONLY

Deadtime falling value and sign is read-only.

HRTIM Deadtime Falling Sign**HRTIM_TIMDEADTIME_FALLINGSIGN_POSITIVE**

Positive deadtime on falling edge.

HRTIM_TIMDEADTIME_FALLINGSIGN_NEGATIVE

Negative deadtime on falling edge.

HRTIM Deadtime Falling Sign Lock**HRTIM_TIMDEADTIME_FALLINGSIGNLOCK_WRITE**

Deadtime falling sign is writeable.

HRTIM_TIMDEADTIME_FALLINGSIGNLOCK_READONLY

Deadtime falling sign is read-only.

HRTIM Deadtime Prescaler Ratio**HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL8** $fDTG = fHRTIM * 8.$ **HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL4** $fDTG = fHRTIM * 4.$ **HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL2** $fDTG = fHRTIM * 2.$ **HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV1** $fDTG = fHRTIM.$ **HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV2** $fDTG = fHRTIM / 2.$ **HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV4** $fDTG = fHRTIM / 4.$ **HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV8** $fDTG = fHRTIM / 8.$ **HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV16** $fDTG = fHRTIM / 16.$ ***HRTIM Deadtime Rising Lock*****HRTIM_TIMDEADTIME_RISINGLOCK_WRITE**

Deadtime rising value and sign is writeable.

HRTIM_TIMDEADTIME_RISINGLOCK_READONLY

Deadtime rising value and sign is read-only.

HRTIM Deadtime Rising Sign**HRTIM_TIMDEADTIME_RISINGSIGN_POSITIVE**

Positive deadtime on rising edge.

HRTIM_TIMDEADTIME_RISINGSIGN_NEGATIVE

Negative deadtime on rising edge.

HRTIM Deadtime Rising Sign Lock**HRTIM_TIMDEADTIME_RISINGSIGNLOCK_WRITE**

Deadtime rising sign is writable.

HRTIM_TIMDEADTIME_RISINGSIGNLOCK_READONLY

Deadtime rising sign is read-only.

HRTIM Exported Macros**_HAL_HRTIM_RESET_HANDLE_STATE****Description:**

- Reset HRTIM handle state.

Parameters:

- __HANDLE__: HRTIM handle.

Return value:

- None

_HAL_HRTIM_ENABLE**Description:**

- Enables or disables the timer counter(s)

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMERS__: timers to enable/disable This parameter can be any combinations of the following values:
 - HRTIM_TIMERID_MASTER: Master timer identifier
 - HRTIM_TIMERID_TIMER_A: Timer A identifier
 - HRTIM_TIMERID_TIMER_B: Timer B identifier
 - HRTIM_TIMERID_TIMER_C: Timer C identifier
 - HRTIM_TIMERID_TIMER_D: Timer D identifier
 - HRTIM_TIMERID_TIMER_E: Timer E identifier

Return value:

- None

HRTIM_TAOEN_MASK**HRTIM_TBOEN_MASK****HRTIM_TCOEN_MASK****HRTIM_TDOEN_MASK****HRTIM_TEOEN_MASK****_HAL_HRTIM_DISABLE****_HAL_HRTIM_ENABLE_IT****Description:**

- Enables or disables the specified HRTIM common interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.

- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - HRTIM_IT_FLT1: Fault 1 interrupt enable
 - HRTIM_IT_FLT2: Fault 2 interrupt enable
 - HRTIM_IT_FLT3: Fault 3 interrupt enable
 - HRTIM_IT_FLT4: Fault 4 interrupt enable
 - HRTIM_IT_FLT5: Fault 5 interrupt enable
 - HRTIM_IT_SYSFLT: System Fault interrupt enable
 - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

- None

[__HAL_HRTIM_ENABLE_IT](#)**Description:**

- Enables or disables the specified HRTIM common interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - HRTIM_IT_FLT1: Fault 1 interrupt enable
 - HRTIM_IT_FLT2: Fault 2 interrupt enable
 - HRTIM_IT_FLT3: Fault 3 interrupt enable
 - HRTIM_IT_FLT4: Fault 4 interrupt enable
 - HRTIM_IT_FLT5: Fault 5 interrupt enable
 - HRTIM_IT_SYSFLT: System Fault interrupt enable
 - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

- None

[__HAL_HRTIM_DISABLE_IT](#)[__HAL_HRTIM_DISABLE_IT](#)[__HAL_HRTIM_MASTER_ENABLE_IT](#)**Description:**

- Enables or disables the specified HRTIM Master timer interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
 - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
 - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
 - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
 - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
 - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
 - HRTIM_MASTER_IT_MUPD: Master update interrupt enable

Return value:

- None

[__HAL_HRTIM_MASTER_DISABLE_IT](#)

[__HAL_HRTIM_TIMER_ENABLE_IT](#)

Description:

- Enables or disables the specified HRTIM Timerx interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt enable
 - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt enable
 - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt enable
 - HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt enable
 - HRTIM_TIM_IT REP: Timer repetition interrupt enable
 - HRTIM_TIM_IT_UPD: Timer update interrupt enable
 - HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt enable
 - HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt enable
 - HRTIM_TIM_IT_SET1: Timer output 1 set interrupt enable
 - HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt enable
 - HRTIM_TIM_IT_SET2: Timer output 2 set interrupt enable
 - HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt enable
 - HRTIM_TIM_IT_RST: Timer reset interrupt enable
 - HRTIM_TIM_IT_DLYPRT: Timer delay protection interrupt enable

Return value:

- None

[__HAL_HRTIM_TIMER_DISABLE_IT](#)

[__HAL_HRTIM_GET_ITSTATUS](#)

Description:

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

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
 - HRTIM_IT_FLT1: Fault 1 interrupt enable
 - HRTIM_IT_FLT2: Fault 2 interrupt enable
 - HRTIM_IT_FLT3: Fault 3 enable
 - HRTIM_IT_FLT4: Fault 4 enable
 - HRTIM_IT_FLT5: Fault 5 enable
 - HRTIM_IT_SYSFLT: System Fault interrupt enable
 - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

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

__HAL_HRTIM_MASTER_GET_ITSTATUS

Description:

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

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
 - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
 - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
 - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
 - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
 - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
 - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
 - HRTIM_MASTER_IT_MUPD: Master update interrupt enable

Return value:

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

__HAL_HRTIM_TIMER_GET_ITSTATUS

Description:

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

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
 - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
 - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
 - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
 - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
 - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
 - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
 - HRTIM_MASTER_IT_MUPD: Master update interrupt enable
 - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt enable
 - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt enable
 - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt enable
 - HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt enable
 - HRTIM_TIM_IT REP: Timer repetition interrupt enable
 - HRTIM_TIM_IT_UPD: Timer update interrupt enable
 - HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt enable
 - HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt enable
 - HRTIM_TIM_IT_SET1: Timer output 1 set interrupt enable
 - HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt enable
 - HRTIM_TIM_IT_SET2: Timer output 2 set interrupt enable
 - HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt enable
 - HRTIM_TIM_IT_RST: Timer reset interrupt enable
 - HRTIM_TIM_IT_DLYPRT: Timer delay protection interrupt enable

Return value:

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

[_HAL_HRTIM_CLEAR_IT](#)

Description:

- Clears the specified HRTIM common pending flag.

Parameters:

- `__HANDLE__`: specifies the HRTIM Handle.
- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - `HRTIM_IT_FLT1`: Fault 1 interrupt clear flag
 - `HRTIM_IT_FLT2`: Fault 2 interrupt clear flag
 - `HRTIM_IT_FLT3`: Fault 3 clear flag
 - `HRTIM_IT_FLT4`: Fault 4 clear flag
 - `HRTIM_IT_FLT5`: Fault 5 clear flag
 - `HRTIM_IT_SYSFLT`: System Fault interrupt clear flag
 - `HRTIM_IT_BMPER`: Burst mode period interrupt clear flag

Return value:

- None

[_HAL_HRTIM_MASTER_CLEAR_IT](#)

Description:

- Clears the specified HRTIM Master pending flag.

Parameters:

- `__HANDLE__`: specifies the HRTIM Handle.
- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - `HRTIM_MASTER_IT_MCMP1`: Master compare 1 interrupt clear flag
 - `HRTIM_MASTER_IT_MCMP2`: Master compare 2 interrupt clear flag
 - `HRTIM_MASTER_IT_MCMP3`: Master compare 3 interrupt clear flag
 - `HRTIM_MASTER_IT_MCMP4`: Master compare 4 interrupt clear flag
 - `HRTIM_MASTER_IT_MREP`: Master Repetition interrupt clear flag
 - `HRTIM_MASTER_IT_SYNC`: Synchronization input interrupt clear flag
 - `HRTIM_MASTER_IT_MUPD`: Master update interrupt clear flag

Return value:

- None

[_HAL_HRTIM_TIMER_CLEAR_IT](#)

Description:

- Clears the specified HRTIM Timerx pending flag.

Parameters:

- `__HANDLE__`: specifies the HRTIM Handle.
- `__TIMER__`: specified the timing unit (Timer A to E)
- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - `HRTIM_TIM_IT_CMP1`: Timer compare 1 interrupt clear flag
 - `HRTIM_TIM_IT_CMP2`: Timer compare 2 interrupt clear flag
 - `HRTIM_TIM_IT_CMP3`: Timer compare 3 interrupt clear flag
 - `HRTIM_TIM_IT_CMP4`: Timer compare 4 interrupt clear flag

- HRTIM_TIM_IT REP: Timer repetition interrupt clear flag
- HRTIM_TIM_IT_UPD: Timer update interrupt clear flag
- HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt clear flag
- HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt clear flag
- HRTIM_TIM_IT_SET1: Timer output 1 set interrupt clear flag
- HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt clear flag
- HRTIM_TIM_IT_SET2: Timer output 2 set interrupt clear flag
- HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt clear flag
- HRTIM_TIM_IT_RST: Timer reset interrupt clear flag
- HRTIM_TIM_IT_DLYPRT: Timer output 1 delay protection interrupt clear flag

Return value:

- None

[__HAL_HRTIM_MASTER_ENABLE_DMA](#)**Description:**

- Enables or disables the specified HRTIM Master timer DMA requests.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __DMA__: specifies the DMA request to enable or disable. This parameter can be one of the following values:
 - HRTIM_MASTER_DMA_MCMP1: Master compare 1 DMA request enable
 - HRTIM_MASTER_DMA_MCMP2: Master compare 2 DMA request enable
 - HRTIM_MASTER_DMA_MCMP3: Master compare 3 DMA request enable
 - HRTIM_MASTER_DMA_MCMP4: Master compare 4 DMA request enable
 - HRTIM_MASTER_DMA_MREP: Master Repetition DMA request enable
 - HRTIM_MASTER_DMA_SYNC: Synchronization input DMA request enable
 - HRTIM_MASTER_DMA_MUPD: Master update DMA request enable

Return value:

- None

[__HAL_HRTIM_MASTER_DISABLE_DMA](#)[__HAL_HRTIM_TIMER_ENABLE_DMA](#)**Description:**

- Enables or disables the specified HRTIM Timerx DMA requests.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __DMA__: specifies the DMA request to enable or disable. This parameter can be one of the following values:
 - HRTIM_TIM_DMA_CMP1: Timer compare 1 DMA request enable
 - HRTIM_TIM_DMA_CMP2: Timer compare 2 DMA request enable
 - HRTIM_TIM_DMA_CMP3: Timer compare 3 DMA request enable
 - HRTIM_TIM_DMA_CMP4: Timer compare 4 DMA request enable
 - HRTIM_TIM_DMA_REP: Timer repetition DMA request enable
 - HRTIM_TIM_DMA_UPD: Timer update DMA request enable
 - HRTIM_TIM_DMA_CPT1: Timer capture 1 DMA request enable
 - HRTIM_TIM_DMA_CPT2: Timer capture 2 DMA request enable

- HRTIM_TIM_DMA_SET1: Timer output 1 set DMA request enable
- HRTIM_TIM_DMA_RST1: Timer output 1 reset DMA request enable
- HRTIM_TIM_DMA_SET2: Timer output 2 set DMA request enable
- HRTIM_TIM_DMA_RST2: Timer output 2 reset DMA request enable
- HRTIM_TIM_DMA_RST: Timer reset DMA request enable
- HRTIM_TIM_DMA_DLYPRT: Timer delay protection DMA request enable

Return value:

- None

[_HAL_HRTIM_TIMER_DISABLE_DMA](#)

[_HAL_HRTIM_GET_FLAG](#)

[_HAL_HRTIM_CLEAR_FLAG](#)

[_HAL_HRTIM_MASTER_GET_FLAG](#)

[_HAL_HRTIM_MASTER_CLEAR_FLAG](#)

[_HAL_HRTIM_TIMER_GET_FLAG](#)

[_HAL_HRTIM_TIMER_CLEAR_FLAG](#)

[_HAL_HRTIM_SETCOUNTER](#)

Description:

- Sets the HRTIM timer Counter Register value on runtime.

Parameters:

- HANDLE: HRTIM Handle.
- TIMER: HRTIM timer This parameter can be one of the following values:
 - 0x5 for master timer
 - 0x0 to 0x4 for timers A to E
- COUNTER: specifies the Counter Register new value.

Return value:

- None

[_HAL_HRTIM_GETCOUNTER](#)

Description:

- Gets the HRTIM timer Counter Register value on runtime.

Parameters:

- HANDLE: HRTIM Handle.
- TIMER: HRTIM timer This parameter can be one of the following values:
 - 0x5 for master timer
 - 0x0 to 0x4 for timers A to E

Return value:

- HRTIM: timer Counter Register value

[_HAL_HRTIM_SETPERIOD](#)

Description:

- Sets the HRTIM timer Period value on runtime.

Parameters:

- `__HANDLE__`: HRTIM Handle.
- `__TIMER__`: HRTIM timer This parameter can be one of the following values:
 - 0x5 for master timer
 - 0x0 to 0x4 for timers A to E
- `__PERIOD__`: specifies the Period Register new value.

Return value:

- None

[`__HAL_HRTIM_GETPERIOD`](#)**Description:**

- Gets the HRTIM timer Period Register value on runtime.

Parameters:

- `__HANDLE__`: HRTIM Handle.
- `__TIMER__`: HRTIM timer This parameter can be one of the following values:
 - 0x5 for master timer
 - 0x0 to 0x4 for timers A to E

Return value:

- timer: Period Register

[`__HAL_HRTIM_SETCLOCKPRESCALER`](#)**Description:**

- Sets the HRTIM timer clock prescaler value on runtime.

Parameters:

- `__HANDLE__`: HRTIM Handle.
- `__TIMER__`: HRTIM timer This parameter can be one of the following values:
 - 0x5 for master timer
 - 0x0 to 0x4 for timers A to E
- `__PRESCALER__`: specifies the clock prescaler new value. This parameter can be one of the following values:
 - `HRTIM_PRESCALERRATIO_MUL32`: fHRCK: 4.608 GHz - Resolution: 217 ps - Min PWM frequency: 70.3 kHz (fHRTIM=144MHz)
 - `HRTIM_PRESCALERRATIO_MUL16`: fHRCK: 2.304 GHz - Resolution: 434 ps - Min PWM frequency: 35.1 KHz (fHRTIM=144MHz)
 - `HRTIM_PRESCALERRATIO_MUL8`: fHRCK: 1.152 GHz - Resolution: 868 ps - Min PWM frequency: 17.6 kHz (fHRTIM=144MHz)
 - `HRTIM_PRESCALERRATIO_MUL4`: fHRCK: 576 MHz - Resolution: 1.73 ns - Min PWM frequency: 8.8 kHz (fHRTIM=144MHz)
 - `HRTIM_PRESCALERRATIO_MUL2`: fHRCK: 288 MHz - Resolution: 3.47 ns - Min PWM frequency: 4.4 kHz (fHRTIM=144MHz)
 - `HRTIM_PRESCALERRATIO_DIV1`: fHRCK: 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)
 - `HRTIM_PRESCALERRATIO_DIV2`: fHRCK: 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)
 - `HRTIM_PRESCALERRATIO_DIV4`: fHRCK: 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

Return value:

- None

__HAL_HRTIM_GETCLOCKPRESCALER

Description:

- Gets the HRTIM timer clock prescaler value on runtime.

Parameters:

- `__HANDLE__`: HRTIM Handle.
- `__TIMER__`: HRTIM timer This parameter can be one of the following values:
 - 0x5 for master timer
 - 0x0 to 0x4 for timers A to E

Return value:

- timer: clock prescaler value

__HAL_HRTIM_SETCOMPARE

Description:

- Sets the HRTIM timer Compare Register value on runtime.

Parameters:

- `__HANDLE__`: HRTIM Handle.
- `__TIMER__`: HRTIM timer This parameter can be one of the following values:
 - 0x0 to 0x4 for timers A to E
- `__COMPAREUNIT__`: timer compare unit This parameter can be one of the following values:
 - `HRTIM_COMPAREUNIT_1`: Compare unit 1
 - `HRTIM_COMPAREUNIT_2`: Compare unit 2
 - `HRTIM_COMPAREUNIT_3`: Compare unit 3
 - `HRTIM_COMPAREUNIT_4`: Compare unit 4
- `__COMPARE__`: specifies the Compare new value.

Return value:

- None

__HAL_HRTIM_GETCOMPARE

Description:

- Gets the HRTIM timer Compare Register value on runtime.

Parameters:

- `__HANDLE__`: HRTIM Handle.
- `__TIMER__`: HRTIM timer This parameter can be one of the following values:
 - 0x0 to 0x4 for timers A to E
- `__COMPAREUNIT__`: timer compare unit This parameter can be one of the following values:
 - `HRTIM_COMPAREUNIT_1`: Compare unit 1
 - `HRTIM_COMPAREUNIT_2`: Compare unit 2
 - `HRTIM_COMPAREUNIT_3`: Compare unit 3
 - `HRTIM_COMPAREUNIT_4`: Compare unit 4

Return value:

- Compare: value

HRTIM External Event Channels

HRTIM_EVENT_NONE

Undefined event channel.

HRTIM_EVENT_1

External event channel 1 identifier.

HRTIM_EVENT_2

External event channel 2 identifier.

HRTIM_EVENT_3

External event channel 3 identifier.

HRTIM_EVENT_4

External event channel 4 identifier.

HRTIM_EVENT_5

External event channel 5 identifier.

HRTIM_EVENT_6

External event channel 6 identifier.

HRTIM_EVENT_7

External event channel 7 identifier.

HRTIM_EVENT_8

External event channel 8 identifier.

HRTIM_EVENT_9

External event channel 9 identifier.

HRTIM_EVENT_10

External event channel 10 identifier.

HRTIM External Event Fast Mode**HRTIM_EVENTFASTMODE_DISABLE**

External Event is acting asynchronously on outputs (low latency mode).

HRTIM_EVENTFASTMODE_ENABLE

External Event is re-synchronized by the HRTIM logic before acting on outputs.

HRTIM External Event Filter**HRTIM_EVENTFILTER_NONE**

Filter disabled.

HRTIM_EVENTFILTER_1

fSAMPLING= fHRTIM, N=2.

HRTIM_EVENTFILTER_2

fSAMPLING= fHRTIM, N=4.

HRTIM_EVENTFILTER_3

fSAMPLING= fHRTIM, N=8.

HRTIM_EVENTFILTER_4

fSAMPLING= fEEVS/2, N=6.

HRTIM_EVENTFILTER_5

fSAMPLING= fEEVS/2, N=8.

HRTIM_EVENTFILTER_6

fSAMPLING= fEEVS/4, N=6.

HRTIM_EVENTFILTER_7

fSAMPLING= fEEVS/4, N=8.

HRTIM_EVENTFILTER_8

fSAMPLING= fEEVS/8, N=6.

HRTIM_EVENTFILTER_9

fSAMPLING= fEEVS/8, N=8.

HRTIM_EVENTFILTER_10

fSAMPLING= fEEVS/16, N=5.

HRTIM_EVENTFILTER_11

fSAMPLING= fEEVS/16, N=6.

HRTIM_EVENTFILTER_12

fSAMPLING= fEEVS/16, N=8.

HRTIM_EVENTFILTER_13

fSAMPLING= fEEVS/32, N=5.

HRTIM_EVENTFILTER_14

fSAMPLING= fEEVS/32, N=6.

HRTIM_EVENTFILTER_15

fSAMPLING= fEEVS/32, N=8.

HRTIM External Event Polarity**HRTIM_EVENTPOLARITY_HIGH**

External event is active high.

HRTIM_EVENTPOLARITY_LOW

External event is active low.

HRTIM External Event Prescaler**HRTIM_EVENTPRESCALER_DIV1**

fEEVS=fHRTIM.

HRTIM_EVENTPRESCALER_DIV2

fEEVS=fHRTIM / 2.

HRTIM_EVENTPRESCALER_DIV4

fEEVS=fHRTIM / 4.

HRTIM_EVENTPRESCALER_DIV8

fEEVS=fHRTIM / 8.

HRTIM External Event Sensitivity**HRTIM_EVENTSensitivity_LEVEL**

External event is active on level.

HRTIM_EVENTSensitivity_RISINGEDGE

External event is active on Rising edge.

HRTIM_EVENTSensitivity_FALLINGEDGE

External event is active on Falling edge.

HRTIM_EVENTSensitivity_BOTHEDGES

External event is active on Rising and Falling edges.

HRTIM External Event Sources**HRTIM_EVENTSRC_1**

External event source 1.

HRTIM_EVENTSRC_2

External event source 2.

HRTIM_EVENTSRC_3

External event source 3.

HRTIM_EVENTSRC_4

External event source 4.

HRTIM External Fault Prescaler**HRTIM_FAULTPRESCALER_DIV1**

fFLTS=fHRTIM.

HRTIM_FAULTPRESCALER_DIV2

fFLTS=fHRTIM / 2.

HRTIM_FAULTPRESCALER_DIV4

fFLTS=fHRTIM / 4.

HRTIM_FAULTPRESCALER_DIV8

fFLTS=fHRTIM / 8.

HRTIM Fault Channel**HRTIM_FAULT_1**

Fault channel 1 identifier.

HRTIM_FAULT_2

Fault channel 2 identifier.

HRTIM_FAULT_3

Fault channel 3 identifier.

HRTIM_FAULT_4

Fault channel 4 identifier.

HRTIM_FAULT_5

Fault channel 5 identifier.

HRTIM Fault Filter**HRTIM_FAULTFILTER_NONE**

Filter disabled.

HRTIM_FAULTFILTER_1

fSAMPLING= fHRTIM, N=2.

HRTIM_FAULTFILTER_2

fSAMPLING= fHRTIM, N=4.

HRTIM_FAULTFILTER_3

fSAMPLING= fHRTIM, N=8.

HRTIM_FAULTFILTER_4

fSAMPLING= fFLTS/2, N=6.

HRTIM_FAULTFILTER_5

fSAMPLING= fFLTS/2, N=8.

HRTIM_FAULTFILTER_6

fSAMPLING= fFLTS/4, N=6.

HRTIM_FAULTFILTER_7

fSAMPLING= fFLTS/4, N=8.

HRTIM_FAULTFILTER_8

fSAMPLING= fFLTS/8, N=6.

HRTIM_FAULTFILTER_9

fSAMPLING= fFLTS/8, N=8.

HRTIM_FAULTFILTER_10

fSAMPLING= fFLTS/16, N=5.

HRTIM_FAULTFILTER_11

fSAMPLING= fFLTS/16, N=6.

HRTIM_FAULTFILTER_12

fSAMPLING= fFLTS/16, N=8.

HRTIM_FAULTFILTER_13

fSAMPLING= fFLTS/32, N=5.

HRTIM_FAULTFILTER_14

fSAMPLING= fFLTS/32, N=6.

HRTIM_FAULTFILTER_15

fSAMPLING= fFLTS/32, N=8.

HRTIM Fault Lock**HRTIM_FAULTLOCK_READWRITE**

Fault settings bits are read/write.

HRTIM_FAULTLOCK_READONLY

Fault settings bits are read only.

HRTIM Fault Mode Control**HRTIM_FAULTMODECTL_DISABLED**

Fault channel is disabled.

HRTIM_FAULTMODECTL_ENABLED

Fault channel is enabled.

HRTIM Fault Polarity**HRTIM_FAULTPOLARITY_LOW**

Fault input is active low.

HRTIM_FAULTPOLARITY_HIGH

Fault input is active high.

HRTIM Fault Sources**HRTIM_FAULTSOURCE_DIGITALINPUT**

Fault input is FLT input pin.

HRTIM_FAULTSOURCE_INTERNAL

Fault input is FLT_Int signal (e.g. internal comparator).

HRTIM Half Mode Enable**HRTIM_HALFMODE_DISABLED**

Half mode is disabled.

HRTIM_HALFMODE_ENABLED

Half mode is enabled.

HRTIM Idle Push Pull Status**HRTIM_PUSHPULL_IDLESTATUS_OUTPUT1**

Protection occurred when the output 1 was active and output 2 forced inactive.

HRTIM_PUSHPULL_IDLESTATUS_OUTPUT2

Protection occurred when the output 2 was active and output 1 forced inactive.

HRTIM Master DMA Request Enable**HRTIM_MASTER_DMA_NONE**

No DMA request enable.

HRTIM_MASTER_DMA_MCMP1

Master compare 1 DMA request enable.

HRTIM_MASTER_DMA_MCMP2

Master compare 2 DMA request enable.

HRTIM_MASTER_DMA_MCMP3

Master compare 3 DMA request enable.

HRTIM_MASTER_DMA_MCMP4

Master compare 4 DMA request enable.

HRTIM_MASTER_DMA_MREP

Master Repetition DMA request enable.

HRTIM_MASTER_DMA_SYNC

Synchronization input DMA request enable.

HRTIM_MASTER_DMA_MUPD

Master update DMA request enable.

HRTIM Master Interrupt Enable**HRTIM_MASTER_IT_NONE**

No interrupt enabled.

HRTIM_MASTER_IT_MCMP1

Master compare 1 interrupt enable.

HRTIM_MASTER_IT_MCMP2

Master compare 2 interrupt enable.

HRTIM_MASTER_IT_MCMP3

Master compare 3 interrupt enable.

HRTIM_MASTER_IT_MCMP4

Master compare 4 interrupt enable.

HRTIM_MASTER_IT_MREP

Master Repetition interrupt enable.

HRTIM_MASTER_IT_SYNC

Synchronization input interrupt enable.

HRTIM_MASTER_IT_MUPD

Master update interrupt enable.

HRTIM Master Interrupt Flag**HRTIM_MASTER_FLAG_MCMP1**

Master compare 1 interrupt flag.

HRTIM_MASTER_FLAG_MCMP2

Master compare 2 interrupt flag.

HRTIM_MASTER_FLAG_MCMP3

Master compare 3 interrupt flag.

HRTIM_MASTER_FLAG_MCMP4

Master compare 4 interrupt flag.

HRTIM_MASTER_FLAG_MREP

Master Repetition interrupt flag.

HRTIM_MASTER_FLAG_SYNC

Synchronization input interrupt flag.

HRTIM_MASTER_FLAG_MUPD

Master update interrupt flag.

HRTIM Max Timer**MAX_HRTIM_TIMER*****HRTIM Mode***

HRTIM_MODE_CONTINUOUS

The timer operates in continuous (free-running) mode.

HRTIM_MODE_SINGLESHOT

The timer operates in non retriggerable single-shot mode.

HRTIM_MODE_SINGLESHOT_RETRIGGERABLE

The timer operates in retriggerable single-shot mode.

HRTIM Output Burst Mode Entry Delayed**HRTIM_OUTPUTBURSTMODEENTRY_REGULAR**

The programmed Idle state is applied immediately to the Output.

HRTIM_OUTPUTBURSTMODEENTRY_DELAYED

Deadtime is inserted on output before entering the idle mode.

HRTIM Output Chopper Mode Enable**HRTIM_OUTPUTCHOPPERMODE_DISABLED**

Output signal is not altered.

HRTIM_OUTPUTCHOPPERMODE_ENABLED

Output signal is chopped by a carrier signal.

HRTIM Output FAULT Level**HRTIM_OUTPUTFAULTLEVEL_NONE**

The output is not affected by the fault input.

HRTIM_OUTPUTFAULTLEVEL_ACTIVE

Output at active level when in FAULT state.

HRTIM_OUTPUTFAULTLEVEL_INACTIVE

Output at inactive level when in FAULT state.

HRTIM_OUTPUTFAULTLEVEL_HIGHZ

Output is tri-stated when in FAULT state.

HRTIM Output IDLE Level**HRTIM_OUTPUTIDLELEVEL_INACTIVE**

Output at inactive level when in IDLE state.

HRTIM_OUTPUTIDLELEVEL_ACTIVE

Output at active level when in IDLE state.

HRTIM Output Idle Mode**HRTIM_OUTPUTIDLEMODE_NONE**

The output is not affected by the burst mode operation.

HRTIM_OUTPUTIDLEMODE_IDLE

The output is in idle state when requested by the burst mode controller.

HRTIM Output Level

HRTIM_OUTPUTLEVEL_ACTIVE

Forces the output to its active state.

HRTIM_OUTPUTLEVEL_INACTIVE

Forces the output to its inactive state.

HRTIM Output Polarity**HRTIM_OUTPUTPOLARITY_HIGH**

Output is active HIGH.

HRTIM_OUTPUTPOLARITY_LOW

Output is active LOW.

HRTIM Output Reset Source**HRTIM_OUTPUTRESET_NONE**

Reset the output reset crossbar.

HRTIM_OUTPUTRESET_RESYNC

Timer reset event coming solely from software or SYNC input forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMPER

Timer period event forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMCMP1

Timer compare 1 event forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMCMP2

Timer compare 2 event forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMCMP3

Timer compare 3 event forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMCMP4

Timer compare 4 event forces the output to its inactive state.

HRTIM_OUTPUTRESET_MASTERPER

The master timer period event forces the output to its inactive state.

HRTIM_OUTPUTRESET_MASTERCMP1

Master Timer compare 1 event forces the output to its inactive state.

HRTIM_OUTPUTRESET_MASTERCMP2

Master Timer compare 2 event forces the output to its inactive state.

HRTIM_OUTPUTRESET_MASTERCMP3

Master Timer compare 3 event forces the output to its inactive state.

HRTIM_OUTPUTRESET_MASTERCMP4

Master Timer compare 4 event forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_1

Timer event 1 forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_2

Timer event 2 forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_3

Timer event 3 forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_4

Timer event 4 forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_5

Timer event 5 forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_6

Timer event 6 forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_7

Timer event 7 forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_8

Timer event 8 forces the output to its inactive state.

HRTIM_OUTPUTRESET_TIMEV_9

Timer event 9 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_1

External event 1 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_2

External event 2 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_3

External event 3 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_4

External event 4 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_5

External event 5 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_6

External event 6 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_7

External event 7 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_8

External event 8 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_9

External event 9 forces the output to its inactive state.

HRTIM_OUTPUTRESET_EEV_10

External event 10 forces the output to its inactive state.

HRTIM_OUTPUTRESET_UPDATE

Timer register update event forces the output to its inactive state.

HRTIM Output Set Source

HRTIM_OUTPUTSET_NONE

Reset the output set crossbar.

HRTIM_OUTPUTSET_RESYNC

Timer reset event coming solely from software or SYNC input forces the output to its active state.

HRTIM_OUTPUTSET_TIMPER

Timer period event forces the output to its active state.

HRTIM_OUTPUTSET_TIMCMP1

Timer compare 1 event forces the output to its active state.

HRTIM_OUTPUTSET_TIMCMP2

Timer compare 2 event forces the output to its active state.

HRTIM_OUTPUTSET_TIMCMP3

Timer compare 3 event forces the output to its active state.

HRTIM_OUTPUTSET_TIMCMP4

Timer compare 4 event forces the output to its active state.

HRTIM_OUTPUTSET_MASTERPER

The master timer period event forces the output to its active state.

HRTIM_OUTPUTSET_MASTERCMP1

Master Timer compare 1 event forces the output to its active state.

HRTIM_OUTPUTSET_MASTERCMP2

Master Timer compare 2 event forces the output to its active state.

HRTIM_OUTPUTSET_MASTERCMP3

Master Timer compare 3 event forces the output to its active state.

HRTIM_OUTPUTSET_MASTERCMP4

Master Timer compare 4 event forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_1

Timer event 1 forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_2

Timer event 2 forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_3

Timer event 3 forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_4

Timer event 4 forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_5

Timer event 5 forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_6

Timer event 6 forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_7

Timer event 7 forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_8

Timer event 8 forces the output to its active state.

HRTIM_OUTPUTSET_TIMEV_9

Timer event 9 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_1

External event 1 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_2

External event 2 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_3

External event 3 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_4

External event 4 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_5

External event 5 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_6

External event 6 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_7

External event 7 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_8

External event 8 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_9

External event 9 forces the output to its active state.

HRTIM_OUTPUTSET_EEV_10

External event 10 forces the output to its active state.

HRTIM_OUTPUTSET_UPDATE

Timer register update event forces the output to its active state.

HRTIM Output State**HRTIM_OUTPUTSTATE_IDLE**

Main operating mode, where the output can take the active or inactive level as programmed in the crossbar unit.

HRTIM_OUTPUTSTATE_RUN

Default operating state (e.g. after an HRTIM reset, when the outputs are disabled by software or during a burst mode operation).

HRTIM_OUTPUTSTATE_FAULT

Safety state, entered in case of a shut-down request on FAULTx inputs.

HRTIM Prescaler Ratio**HRTIM_PRESCALERRATIO_MUL32**

fHRCK: 4.608 GHz - Resolution: 217 ps - Min PWM frequency: 70.3 kHz (fHRTIM=144MHz).

HRTIM_PRESCALERRATIO_MUL16

fHRCK: 2.304 GHz - Resolution: 434 ps - Min PWM frequency: 35.1 KHz (fHRTIM=144MHz).

HRTIM_PRESCALERRATIO_MUL8

fHRCK: 1.152 GHz - Resolution: 868 ps - Min PWM frequency: 17.6 kHz (fHRTIM=144MHz).

HRTIM_PRESCALERRATIO_MUL4

fHRCK: 576 MHz - Resolution: 1.73 ns - Min PWM frequency: 8.8 kHz (fHRTIM=144MHz).

HRTIM_PRESCALERRATIO_MUL2

fHRCK: 288 MHz - Resolution: 3.47 ns - Min PWM frequency: 4.4 kHz (fHRTIM=144MHz).

HRTIM_PRESCALERRATIO_DIV1

fHRCK: 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz).

HRTIM_PRESCALERRATIO_DIV2

fHRCK: 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz).

HRTIM_PRESCALERRATIO_DIV4

fHRCK: 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz).

HRTIM Register Preload Enable**HRTIM_PRELOAD_DISABLED**

Preload disabled: the write access is directly done into the active register.

HRTIM_PRELOAD_ENABLED

Preload enabled: the write access is done into the preload register.

HRTIM Reset On Sync Input Event**HRTIM_SYNCRESET_DISABLED**

Synchronization input event has effect on the timer.

HRTIM_SYNCRESET_ENABLED

Synchronization input event resets the timer.

HRTIM Simple OC Mode**HRTIM_BASICOCMODE_TOGGLE**

Output toggles when the timer counter reaches the compare value.

HRTIM_BASICOCMODE_INACTIVE

Output forced to active level when the timer counter reaches the compare value.

HRTIM_BASICOCMODE_ACTIVE

Output forced to inactive level when the timer counter reaches the compare value.

HRTIM Software Timer Reset**HRTIM_TIMERRESET_MASTER**

Resets the master timer counter.

HRTIM_TIMERRESET_TIMER_A

Resets the timer A counter.

HRTIM_TIMERRESET_TIMER_B

Resets the timer B counter.

HRTIM_TIMERRESET_TIMER_C

Resets the timer C counter.

HRTIM_TIMERRESET_TIMER_D

Resets the timer D counter.

HRTIM_TIMERRESET_TIMER_E

Resets the timer E counter.

HRTIM Software Timer Update**HRTIM_TIMERUPDATE_MASTER**

Forces an immediate transfer from the preload to the active register in the master timer.

HRTIM_TIMERUPDATE_A

Forces an immediate transfer from the preload to the active register in the timer A.

HRTIM_TIMERUPDATE_B

Forces an immediate transfer from the preload to the active register in the timer B.

HRTIM_TIMERUPDATE_C

Forces an immediate transfer from the preload to the active register in the timer C.

HRTIM_TIMERUPDATE_D

Forces an immediate transfer from the preload to the active register in the timer D.

HRTIM_TIMERUPDATE_E

Forces an immediate transfer from the preload to the active register in the timer E.

HRTIM Start On Sync Input Event**HRTIM_SYNCSTART_DISABLED**

Synchronization input event has effect on the timer.

HRTIM_SYNCSTART_ENABLED

Synchronization input event starts the timer.

HRTIM Synchronization Input Source**HRTIM_SYNCINPUTSOURCE_NONE**

disabled. HRTIM is not synchronized and runs in standalone mode.

HRTIM_SYNCINPUTSOURCE_INTERNALEVENT

The HRTIM is synchronized with the on-chip timer.

HRTIM_SYNCINPUTSOURCE_EXTERNALEVENT

A positive pulse on SYNCIN input triggers the HRTIM.

HRTIM Synchronization Options**HRTIM_SYNCOPTION_NONE**

HRTIM instance doesn't handle external synchronization signals (SYNCIN, SYNCOUT).

HRTIM_SYNCOPTION_MASTER

HRTIM instance acts as a MASTER, i.e. generates external synchronization output (SYNCOUT).

HRTIM_SYNCOPTION_SLAVE

HRTIM instance acts as a SLAVE, i.e. it is synchronized by external sources (SYNCIN).

HRTIM Synchronization Output Polarity**HRTIM_SYNCOUTPUTPOLARITY_NONE**

Synchronization output event is disabled.

HRTIM_SYNCOUTPUTPOLARITY_POSITIVE

SCOUT pin has a low idle level and issues a positive pulse of 16 fHRTIM clock cycles length for the synchronization.

HRTIM_SYNCOUTPUTPOLARITY_NEGATIVE

SCOUT pin has a high idle level and issues a negative pulse of 16 fHRTIM clock cycles length for the synchronization.

HRTIM Synchronization Output Source**HRTIM_SYNCOUTPUTSOURCE_MASTER_START**

A pulse is sent on the SYNCOUT output upon master timer start event.

HRTIM_SYNCOUTPUTSOURCE_MASTER_CMP1

A pulse is sent on the SYNCOUT output upon master timer compare 1 event.

HRTIM_SYNCOUTPUTSOURCE_TIMA_START

A pulse is sent on the SYNCOUT output upon timer A start or reset events.

HRTIM_SYNCOUTPUTSOURCE_TIMA_CMP1

A pulse is sent on the SYNCOUT output upon timer A compare 1 event.

HRTIM Timer Burst Mode**HRTIM_TIMERBURSTMODE_MAINTAINCLOCK**

Timer counter clock is maintained and the timer operates normally.

HRTIM_TIMERBURSTMODE_RESETCOUNTER

Timer counter clock is stopped and the counter is reset.

HRTIM Timer Deadtime Insertion**HRTIM_TIMDEADTIMEINSERTION_DISABLED**

Output 1 and output 2 signals are independent.

HRTIM_TIMDEADTIMEINSERTION_ENABLED

Deadtime is inserted between output 1 and output 2.

HRTIM Timer Delayed Protection Mode**HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DISABLED**

No action.

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT1_EEV6

Timers A, B, C: Output 1 delayed Idle on external Event 6.

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT2_EEV6

Timers A, B, C: Output 2 delayed Idle on external Event 6.

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDBOTH_EEV6

Timers A, B, C: Output 1 and output 2 delayed Idle on external Event 6.

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_BALANCED_EEV6

Timers A, B, C: Balanced Idle on external Event 6.

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT1_DEEV7

Timers A, B, C: Output 1 delayed Idle on external Event 7.

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDOUT2_DEEV7

Timers A, B, C: Output 2 delayed Idle on external Event 7.

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DELAYEDBOTH_EEV7

Timers A, B, C: Output 1 and output2 delayed Idle on external Event 7.

HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_BALANCED_EEV7

Timers A, B, C: Balanced Idle on external Event 7.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DISABLED

No action.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT1_EEV8

Timers D, E: Output 1 delayed Idle on external Event 6.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT2_EEV8

Timers D, E: Output 2 delayed Idle on external Event 6.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDBOTH_EEV8

Timers D, E: Output 1 and output 2 delayed Idle on external Event 6.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_BALANCED_EEV8

Timers D, E: Balanced Idle on external Event 6.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT1_DEEV9

Timers D, E: Output 1 delayed Idle on external Event 7.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDOUT2_DEEV9

Timers D, E: Output 2 delayed Idle on external Event 7.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_DELAYEDBOTH_EEV9

Timers D, E: Output 1 and output2 delayed Idle on external Event 7.

HRTIM_TIMER_D_E_DELAYEDPROTECTION_BALANCED_EEV9

Timers D, E: Balanced Idle on external Event 7.

HRTIM Timer External Event Filter**HRTIM_TIMEVENTFILTER_NONE****HRTIM_TIMEVENTFILTER_BLANKINGCMP1**

Blanking from counter reset/roll-over to Compare 1

HRTIM_TIMEVENTFILTER_BLANKINGCMP2

Blanking from counter reset/roll-over to Compare 2

HRTIM_TIMEVENTFILTER_BLANKINGCMP3

Blanking from counter reset/roll-over to Compare 3

HRTIM_TIMEVENTFILTER_BLANKINGCMP4

Blanking from counter reset/roll-over to Compare 4

HRTIM_TIMEVENTFILTER_BLANKINGFLTR1

Blanking from another timing unit: TIMFLTR1 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR2

Blanking from another timing unit: TIMFLTR2 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR3

Blanking from another timing unit: TIMFLTR3 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR4

Blanking from another timing unit: TIMFLTR4 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR5

Blanking from another timing unit: TIMFLTR5 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR6

Blanking from another timing unit: TIMFLTR6 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR7

Blanking from another timing unit: TIMFLTR7 source

HRTIM_TIMEVENTFILTER_BLANKINGFLTR8

Blanking from another timing unit: TIMFLTR8 source

HRTIM_TIMEVENTFILTER_WINDOWINGCMP2

Windowing from counter reset/roll-over to Compare 2

HRTIM_TIMEVENTFILTER_WINDOWINGCMP3

Windowing from counter reset/roll-over to Compare 3

HRTIM_TIMEVENTFILTER_WINDOWINGTIM

Windowing from another timing unit: TIMWIN source

HRTIM Timer External Event Latch**HRTIM_TIMEVENTLATCH_DISABLED**

Event is ignored if it happens during a blank, or passed through during a window.

HRTIM_TIMEVENTLATCH_ENABLED

Event is latched and delayed till the end of the blanking or windowing period.

HRTIM Timer Fault Enabling**HRTIM_TIMFAULTENABLE_NONE**

No fault enabled.

HRTIM_TIMFAULTENABLE_FAULT1

Fault 1 enabled.

HRTIM_TIMFAULTENABLE_FAULT2

Fault 2 enabled.

HRTIM_TIMFAULTENABLE_FAULT3

Fault 3 enabled.

HRTIM_TIMFAULTENABLE_FAULT4

Fault 4 enabled.

HRTIM_TIMFAULTENABLE_FAULT5

Fault 5 enabled.

HRTIM Timer Fault Lock**HRTIM_TIMFAULTLOCK_READWRITE**

Timer fault enabling bits are read/write.

HRTIM_TIMFAULTLOCK_READONLY

Timer fault enabling bits are read only.

HRTIM Timer identifier**HRTIM_TIMERID_MASTER**

Master identifier

HRTIM_TIMERID_TIMER_A

Timer A identifier

HRTIM_TIMERID_TIMER_B

Timer B identifier

HRTIM_TIMERID_TIMER_C

Timer C identifier

HRTIM_TIMERID_TIMER_D

Timer D identifier

HRTIM_TIMERID_TIMER_E

Timer E identifier

IS_HRTIM_TIMERID***HRTIM Timer Index*****HRTIM_TIMERINDEX_TIMER_A**

Index used to access timer A registers

HRTIM_TIMERINDEX_TIMER_B

Index used to access timer B registers

HRTIM_TIMERINDEX_TIMER_C

Index used to access timer C registers

HRTIM_TIMERINDEX_TIMER_D

Index used to access timer D registers

HRTIM_TIMERINDEX_TIMER_E

Index used to access timer E registers

HRTIM_TIMERINDEX_MASTER

Index used to access master registers

HRTIM_TIMERINDEX_COMMON

Index used to access HRTIM common registers

HRTIM Timer Output

HRTIM_OUTPUT_TA1

Timer A - Ouput 1 identifier.

HRTIM_OUTPUT_TA2

Timer A - Ouput 2 identifier.

HRTIM_OUTPUT_TB1

Timer B - Ouput 1 identifier.

HRTIM_OUTPUT_TB2

Timer B - Ouput 2 identifier.

HRTIM_OUTPUT_TC1

Timer C - Ouput 1 identifier.

HRTIM_OUTPUT_TC2

Timer C - Ouput 2 identifier.

HRTIM_OUTPUT_TD1

Timer D - Ouput 1 identifier.

HRTIM_OUTPUT_TD2

Timer D - Ouput 2 identifier.

HRTIM_OUTPUT_TE1

Timer E - Ouput 1 identifier.

HRTIM_OUTPUT_TE2

Timer E - Ouput 2 identifier.

HRTIM Timer Push Pull Mode**HRTIM_TIMPUSHPULLMODE_DISABLED**

Push-Pull mode disabled.

HRTIM_TIMPUSHPULLMODE_ENABLED

Push-Pull mode enabled.

HRTIM Timer Repetition Update**HRTIM_UPDATEONREPETITION_DISABLED**

Update on repetition disabled.

HRTIM_UPDATEONREPETITION_ENABLED

Update on repetition enabled.

HRTIM Timer Reset Trigger**HRTIM_TIMRESETTRIGGER_NONE**

No counter reset trigger.

HRTIM_TIMRESETTRIGGER_UPDATE

The timer counter is reset upon update event.

HRTIM_TIMRESETTRIGGER_CMP2

The timer counter is reset upon Timer Compare 2 event.

HRTIM_TIMRESETTRIGGER_CMP4

The timer counter is reset upon Timer Compare 4 event.

HRTIM_TIMRESETTRIGGER_MASTER_PER

The timer counter is reset upon master timer period event.

HRTIM_TIMRESETTRIGGER_MASTER_CMP1

The timer counter is reset upon master timer Compare 1 event.

HRTIM_TIMRESETTRIGGER_MASTER_CMP2

The timer counter is reset upon master timer Compare 2 event.

HRTIM_TIMRESETTRIGGER_MASTER_CMP3

The timer counter is reset upon master timer Compare 3 event.

HRTIM_TIMRESETTRIGGER_MASTER_CMP4

The timer counter is reset upon master timer Compare 4 event.

HRTIM_TIMRESETTRIGGER_EEV_1

The timer counter is reset upon external event 1.

HRTIM_TIMRESETTRIGGER_EEV_2

The timer counter is reset upon external event 2.

HRTIM_TIMRESETTRIGGER_EEV_3

The timer counter is reset upon external event 3.

HRTIM_TIMRESETTRIGGER_EEV_4

The timer counter is reset upon external event 4.

HRTIM_TIMRESETTRIGGER_EEV_5

The timer counter is reset upon external event 5.

HRTIM_TIMRESETTRIGGER_EEV_6

The timer counter is reset upon external event 6.

HRTIM_TIMRESETTRIGGER_EEV_7

The timer counter is reset upon external event 7.

HRTIM_TIMRESETTRIGGER_EEV_8

The timer counter is reset upon external event 8.

HRTIM_TIMRESETTRIGGER_EEV_9

The timer counter is reset upon external event 9.

HRTIM_TIMRESETTRIGGER_EEV_10

The timer counter is reset upon external event 10.

HRTIM_TIMRESETTRIGGER_OTHER1_CMP1

The timer counter is reset upon other timer Compare 1 event.

HRTIM_TIMRESETTRIGGER_OTHER1_CMP2

The timer counter is reset upon other timer Compare 2 event.

HRTIM_TIMRESETTRIGGER_OTHER1_CMP4

The timer counter is reset upon other timer Compare 4 event.

HRTIM_TIMRESETTRIGGER_OTHER2_CMP1

The timer counter is reset upon other timer Compare 1 event.

HRTIM_TIMRESETTRIGGER_OTHER2_CMP2

The timer counter is reset upon other timer Compare 2 event.

HRTIM_TIMRESETTRIGGER_OTHER2_CMP4

The timer counter is reset upon other timer Compare 4 event.

HRTIM_TIMRESETTRIGGER_OTHER3_CMP1

The timer counter is reset upon other timer Compare 1 event.

HRTIM_TIMRESETTRIGGER_OTHER3_CMP2

The timer counter is reset upon other timer Compare 2 event.

HRTIM_TIMRESETTRIGGER_OTHER3_CMP4

The timer counter is reset upon other timer Compare 4 event.

HRTIM_TIMRESETTRIGGER_OTHER4_CMP1

The timer counter is reset upon other timer Compare 1 event.

HRTIM_TIMRESETTRIGGER_OTHER4_CMP2

The timer counter is reset upon other timer Compare 2 event.

HRTIM_TIMRESETTRIGGER_OTHER4_CMP4

The timer counter is reset upon other timer Compare 4 event.

HRTIM Timer Reset Update**HRTIM_TIMUPDATEONRESET_DISABLED**

Update by timer x reset / roll-over disabled.

HRTIM_TIMUPDATEONRESET_ENABLED

Update by timer x reset / roll-over enabled.

HRTIM Timer Update Trigger**HRTIM_TIMUPDATETRIGGER_NONE**

Register update is disabled.

HRTIM_TIMUPDATETRIGGER_MASTER

Register update is triggered by the master timer update.

HRTIM_TIMUPDATETRIGGER_TIMER_A

Register update is triggered by the timer A update.

HRTIM_TIMUPDATETRIGGER_TIMER_B

Register update is triggered by the timer B update.

HRTIM_TIMUPDATETRIGGER_TIMER_C

Register update is triggered by the timer C update.

HRTIM_TIMUPDATETRIGGER_TIMER_D

Register update is triggered by the timer D update.

HRTIM_TIMUPDATETRIGGER_TIMER_E

Register update is triggered by the timer E update.

HRTIM Timing Unit DMA Request Enable**[HRTIM_TIM_DMA_NONE](#)**

No DMA request enable.

[HRTIM_TIM_DMA_CMP1](#)

Timer compare 1 DMA request enable.

[HRTIM_TIM_DMA_CMP2](#)

Timer compare 2 DMA request enable.

[HRTIM_TIM_DMA_CMP3](#)

Timer compare 3 DMA request enable.

[HRTIM_TIM_DMA_CMP4](#)

Timer compare 4 DMA request enable.

[HRTIM_TIM_DMA REP](#)

Timer repetition DMA request enable.

[HRTIM_TIM_DMA_UPD](#)

Timer update DMA request enable.

[HRTIM_TIM_DMA_CPT1](#)

Timer capture 1 DMA request enable.

[HRTIM_TIM_DMA_CPT2](#)

Timer capture 2 DMA request enable.

[HRTIM_TIM_DMA_SET1](#)

Timer output 1 set DMA request enable.

[HRTIM_TIM_DMA_RST1](#)

Timer output 1 reset DMA request enable.

[HRTIM_TIM_DMA_SET2](#)

Timer output 2 set DMA request enable.

[HRTIM_TIM_DMA_RST2](#)

Timer output 2 reset DMA request enable.

[HRTIM_TIM_DMA_RST](#)

Timer reset DMA request enable.

[HRTIM_TIM_DMA_DLYPRT](#)

Timer delay protection DMA request enable.

HRTIM Timing Unit Interrupt Enable**[HRTIM_TIM_IT_NONE](#)**

No interrupt enabled.

[HRTIM_TIM_IT_CMP1](#)

Timer compare 1 interrupt enable.

[HRTIM_TIM_IT_CMP2](#)

Timer compare 2 interrupt enable.

HRTIM_TIM_IT_CMP3

Timer compare 3 interrupt enable.

HRTIM_TIM_IT_CMP4

Timer compare 4 interrupt enable.

HRTIM_TIM_IT REP

Timer repetition interrupt enable.

HRTIM_TIM_IT_UPD

Timer update interrupt enable.

HRTIM_TIM_IT_CPT1

Timer capture 1 interrupt enable.

HRTIM_TIM_IT_CPT2

Timer capture 2 interrupt enable.

HRTIM_TIM_IT_SET1

Timer output 1 set interrupt enable.

HRTIM_TIM_IT_RST1

Timer output 1 reset interrupt enable.

HRTIM_TIM_IT_SET2

Timer output 2 set interrupt enable.

HRTIM_TIM_IT_RST2

Timer output 2 reset interrupt enable.

HRTIM_TIM_IT_RST

Timer reset interrupt enable.

HRTIM_TIM_IT_DLYPRT

Timer delay protection interrupt enable.

HRTIM Timing Unit Interrupt Flag**HRTIM_TIM_FLAG_CMP1**

Timer compare 1 interrupt flag.

HRTIM_TIM_FLAG_CMP2

Timer compare 2 interrupt flag.

HRTIM_TIM_FLAG_CMP3

Timer compare 3 interrupt flag.

HRTIM_TIM_FLAG_CMP4

Timer compare 4 interrupt flag.

HRTIM_TIM_FLAG REP

Timer repetition interrupt flag.

HRTIM_TIM_FLAG_UPD

Timer update interrupt flag.

HRTIM_TIM_FLAG_CPT1

Timer capture 1 interrupt flag.

HRTIM_TIM_FLAG_CPT2

Timer capture 2 interrupt flag.

HRTIM_TIM_FLAG_SET1

Timer output 1 set interrupt flag.

HRTIM_TIM_FLAG_RST1

Timer output 1 reset interrupt flag.

HRTIM_TIM_FLAG_SET2

Timer output 2 set interrupt flag.

HRTIM_TIM_FLAG_RST2

Timer output 2 reset interrupt flag.

HRTIM_TIM_FLAG_RST

Timer reset interrupt flag.

HRTIM_TIM_FLAG_DLYPRT

Timer delay protection interrupt flag.

HRTIM Update Gating**HRTIM_UPDATEGATING_INDEPENDENT**

Update done independently from the DMA burst transfer completion.

HRTIM_UPDATEGATING_DMABURST

Update done when the DMA burst transfer is completed.

HRTIM_UPDATEGATING_DMABURST_UPDATE

Update done on timer roll-over following a DMA burst transfer completion.

HRTIM_UPDATEGATING_UPDEN1

Slave timer only - Update done on a rising edge of HRTIM update enable input 1.

HRTIM_UPDATEGATING_UPDEN2

Slave timer only - Update done on a rising edge of HRTIM update enable input 2.

HRTIM_UPDATEGATING_UPDEN3

Slave timer only - Update done on a rising edge of HRTIM update enable input 3.

HRTIM_UPDATEGATING_UPDEN1_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 1.

HRTIM_UPDATEGATING_UPDEN2_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 2.

HRTIM_UPDATEGATING_UPDEN3_UPDATE

Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 3.

32 HAL HSEM Generic Driver

32.1 HSEM Firmware driver API description

32.1.1 How to use this driver

1. Take a semaphore In 2-Step mode Using function HAL_HSEM_Take. This function takes as parameters :
 - the semaphore ID from 0 to 31
 - the process ID from 0 to 255
2. Fast Take semaphore In 1-Step mode Using function HAL_HSEM_FastTake. This function takes as parameter :
 - the semaphore ID from 0_ID to 31. Note that the process ID value is implicitly assumed as zero
3. Check if a semaphore is Taken using function HAL_HSEM_IsSemTaken. This function takes as parameter :
 - the semaphore ID from 0_ID to 31
 - It returns 1 if the given semaphore is taken otherwise (Free) zero.
4. Release a semaphore using function with HAL_HSEM_Release. This function takes as parameters :
 - the semaphore ID from 0 to 31
 - the process ID from 0 to 255:
 - Note: If ProcessID and MasterID match, semaphore is freed, and an interrupt may be generated when enabled (notification activated). If ProcessID or MasterID does not match, semaphore remains taken (locked).
5. Release all semaphores at once taken by a given Master using function HAL_HSEM_Release_All This function takes as parameters :
 - the Release Key (value from 0 to 0xFFFF) can be Set or Get respectively by HAL_HSEM_SetClearKey() or HAL_HSEM_GetClearKey functions
 - the Master ID:
 - Note: If the Key and MasterID match, all semaphores taken by the given CPU that corresponds to MasterID will be freed, and an interrupt may be generated when enabled (notification activated). If the Key or the MasterID doesn't match, semaphores remains taken (locked).
6. Semaphores Release all key functions:
 - HAL_HSEM_SetClearKey() to set semaphore release all Key
 - HAL_HSEM_GetClearKey() to get release all Key
7. Semaphores notification functions :
 - HAL_HSEM_ActivateNotification to activate a notification callback on a given semaphores Mask (bitfield). When one or more semaphores defined by the mask are released the callback HAL_HSEM_FreeCallback will be asserted giving as parameters a mask of the released semaphores (bitfield).
 - HAL_HSEM_DeactivateNotification to deactivate the notification of a given semaphores Mask (bitfield).
 - See the description of the macro __HAL_HSEM_SEMID_TO_MASK to check how to calculate a semaphore mask Used by the notification functions

HSEM HAL driver macros list

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

- __HAL_HSEM_SEMID_TO_MASK: Helper macro to convert a Semaphore ID to a Mask.

Example of use :

```
mask = __HAL_HSEM_SEMID_TO_MASK(8) | __HAL_HSEM_SEMID_TO_MASK(21) |  
__HAL_HSEM_SEMID_TO_MASK(25).
```

All next macros take as parameter a semaphore Mask (bitfiled) that can be constructed using `__HAL_HSEM_SEMID_TO_MASK` as the above example.

- `__HAL_HSEM_ENABLE_IT`: Enable the specified semaphores Mask interrupts.
- `__HAL_HSEM_DISABLE_IT`: Disable the specified semaphores Mask interrupts.
- `__HAL_HSEM_GET_IT`: Checks whether the specified semaphore interrupt has occurred or not.
- `__HAL_HSEM_GET_FLAG`: Get the semaphores status release flags.
- `__HAL_HSEM_CLEAR_FLAG`: Clear the semaphores status release flags.

32.1.2 HSEM Take and Release functions

This section provides functions allowing to:

- Take a semaphore with 2 Step method
- Fast Take a semaphore with 1 Step method
- Check semaphore state Taken or not
- Release a semaphore
- Release all semaphore at once

This section contains the following APIs:

- [`HAL_HSEM_Take`](#)
- [`HAL_HSEM_FastTake`](#)
- [`HAL_HSEM_IsSemTaken`](#)
- [`HAL_HSEM_Release`](#)
- [`HAL_HSEM_ReleaseAll`](#)

32.1.3 HSEM Set and Get Key functions

This section provides functions allowing to:

- Set semaphore Key
- Get semaphore Key

This section contains the following APIs:

- [`HAL_HSEM_SetClearKey`](#)
- [`HAL_HSEM_GetClearKey`](#)

32.1.4 HSEM IRQ handler management and Notification functions

This section provides HSEM IRQ handler and Notification function.

This section contains the following APIs:

- [`HAL_HSEM_ActivateNotification`](#)
- [`HAL_HSEM_DeactivateNotification`](#)
- [`HAL_HSEM_IRQHandler`](#)
- [`HAL_HSEM_FreeCallback`](#)

32.1.5 Detailed description of functions

`HAL_HSEM_Take`

Function name

`HAL_StatusTypeDef HAL_HSEM_Take (uint32_t SemID, uint32_t ProcessID)`

Function description

Take a semaphore in 2 Step mode.

Parameters

- **SemID:** semaphore ID from 0 to 31
- **ProcessID:** Process ID from 0 to 255

Return values

- **HAL:** status

HAL_HSEM_FastTake

Function name

HAL_StatusTypeDef HAL_HSEM_FastTake (uint32_t SemID)

Function description

Fast Take a semaphore with 1 Step mode.

Parameters

- **SemID:** semaphore ID from 0 to 31

Return values

- **HAL:** status

HAL_HSEM_IsSemTaken

Function name

uint32_t HAL_HSEM_IsSemTaken (uint32_t SemID)

Function description

Check semaphore state Taken or not.

Parameters

- **SemID:** semaphore ID

Return values

- **HAL:** HSEM state

HAL_HSEM_Release

Function name

void HAL_HSEM_Release (uint32_t SemID, uint32_t ProcessID)

Function description

Release a semaphore.

Parameters

- **SemID:** semaphore ID from 0 to 31
- **ProcessID:** Process ID from 0 to 255

Return values

- **None:**

HAL_HSEM_ReleaseAll

Function name

void HAL_HSEM_ReleaseAll (uint32_t Key, uint32_t MasterID)

Function description

Release All semaphore used by a given Master .

Parameters

- **Key:** Semaphore Key , value from 0 to 0xFFFF
- **MasterID:** MasterID of the CPU that is using semaphores to be Released

Return values

- **None:**

HAL_HSEM_SetClearKey

Function name

void HAL_HSEM_SetClearKey (uint32_t Key)

Function description

Set semaphore Key .

Parameters

- **Key:** Semaphore Key , value from 0 to 0xFFFF

Return values

- **None:**

HAL_HSEM_GetClearKey

Function name

uint32_t HAL_HSEM_GetClearKey (void)

Function description

Get semaphore Key .

Return values

- **Semaphore:** Key , value from 0 to 0xFFFF

HAL_HSEM_ActivateNotification

Function name

void HAL_HSEM_ActivateNotification (uint32_t SemMask)

Function description

Activate Semaphore release Notification for a given Semaphores Mask .

Parameters

- **SemMask:** Mask of Released semaphores

Return values

- **Semaphore:** Key

HAL_HSEM_DeactivateNotification

Function name

void HAL_HSEM_DeactivateNotification (uint32_t SemMask)

Function description

Deactivate Semaphore release Notification for a given Semaphores Mask .

Parameters

- **SemMask:** Mask of Released semaphores

Return values

- **Semaphore:** Key

HAL_HSEM_FreeCallback**Function name****void HAL_HSEM_FreeCallback (uint32_t SemMask)****Function description**

Semaphore Released Callback.

Parameters

- **SemMask:** Mask of Released semaphores

Return values

- **None:**

HAL_HSEM_IRQHandler**Function name****void HAL_HSEM_IRQHandler (void)****Function description**

This function handles HSEM interrupt request.

Return values

- **None:**

32.2 HSEM Firmware driver defines

32.2.1 HSEM

HSEM Exported Macros**_HAL_HSEM_SEMID_TO_MASK****Description:**

- SemID to mask helper Macro.

Parameters:

- **_SEMID_:** semaphore ID from 0 to 31

Return value:

- Semaphore: Mask.

_HAL_HSEM_ENABLE_IT**Description:**

- Enables the specified HSEM interrupts.

Parameters:

- **_SEM_MASK_:** semaphores Mask

Return value:

- None.

_HAL_HSEM_DISABLE_IT**Description:**

- Disables the specified HSEM interrupts.

Parameters:

- `__SEM_MASK__`: semaphores Mask

Return value:

- None.

[`__HAL_HSEM_GET_IT`](#)**Description:**

- Checks whether interrupt has occurred or not for semaphores specified by a mask.

Parameters:

- `__SEM_MASK__`: semaphores Mask

Return value:

- semaphores: Mask : Semaphores where an interrupt occurred.

[`__HAL_HSEM_GET_FLAG`](#)**Description:**

- Get the semaphores release status flags.

Parameters:

- `__SEM_MASK__`: semaphores Mask

Return value:

- semaphores: Mask : Semaphores where Release flags rise.

[`__HAL_HSEM_CLEAR_FLAG`](#)**Description:**

- Clears the HSEM Interrupt flags.

Parameters:

- `__SEM_MASK__`: semaphores Mask

Return value:

- None.

33 HAL I2C Generic Driver

33.1 I2C Firmware driver registers structures

33.1.1 I2C_InitTypeDef

Data Fields

- *uint32_t Timing*
- *uint32_t OwnAddress1*
- *uint32_t AddressingMode*
- *uint32_t DualAddressMode*
- *uint32_t OwnAddress2*
- *uint32_t OwnAddress2Masks*
- *uint32_t GeneralCallMode*
- *uint32_t NoStretchMode*

Field Documentation

- ***uint32_t I2C_InitTypeDef::Timing***

Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual

- ***uint32_t I2C_InitTypeDef::OwnAddress1***

Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.

- ***uint32_t I2C_InitTypeDef::AddressingMode***

Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C Addressing Mode***

- ***uint32_t I2C_InitTypeDef::DualAddressMode***

Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C Dual Addressing Mode***

- ***uint32_t I2C_InitTypeDef::OwnAddress2***

Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.

- ***uint32_t I2C_InitTypeDef::OwnAddress2Masks***

Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of ***I2C Own Address2 Masks***

- ***uint32_t I2C_InitTypeDef::GeneralCallMode***

Specifies if general call mode is selected. This parameter can be a value of ***I2C General Call Addressing Mode***

- ***uint32_t I2C_InitTypeDef::NoStretchMode***

Specifies if nostretch mode is selected. This parameter can be a value of ***I2C No-Stretch Mode***

33.1.2 __I2C_HandleTypeDef

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***

- `uint8_t * pBuffPtr`
- `uint16_t XferSize`
- `__IO uint16_t XferCount`
- `__IO uint32_t XferOptions`
- `__IO uint32_t PreviousState`
- `HAL_StatusTypeDef(* XferISR`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `__IO HAL_I2C_StateTypeDef State`
- `__IO HAL_I2C_ModeTypeDef Mode`
- `__IO uint32_t ErrorCode`
- `__IO uint32_t AddrEventCount`

Field Documentation

- `I2C_HandleTypeDef* __I2C_HandleTypeDef::Instance`
I2C registers base address
- `I2C_InitTypeDef __I2C_HandleTypeDef::Init`
I2C communication parameters
- `uint8_t* __I2C_HandleTypeDef::pBuffPtr`
Pointer to I2C transfer buffer
- `uint16_t __I2C_HandleTypeDef::XferSize`
I2C transfer size
- `__IO uint16_t __I2C_HandleTypeDef::XferCount`
I2C transfer counter
- `__IO uint32_t __I2C_HandleTypeDef::XferOptions`
I2C sequential transfer options, this parameter can be a value of `I2C Sequential Transfer Options`
- `__IO uint32_t __I2C_HandleTypeDef::PreviousState`
I2C communication Previous state
- `HAL_StatusTypeDef(* __I2C_HandleTypeDef::XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint32_t ITSources)`
I2C transfer IRQ handler function pointer
- `DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmatx`
I2C Tx DMA handle parameters
- `DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmarx`
I2C Rx DMA handle parameters
- `HAL_LockTypeDef __I2C_HandleTypeDef::Lock`
I2C locking object
- `__IO HAL_I2C_StateTypeDef __I2C_HandleTypeDef::State`
I2C communication state
- `__IO HAL_I2C_ModeTypeDef __I2C_HandleTypeDef::Mode`
I2C communication mode
- `__IO uint32_t __I2C_HandleTypeDef::ErrorCode`
I2C Error code

- `_IO uint32_t __I2C_HandleTypeDef::AddrEventCount`
I2C Address Event counter

33.2 I2C Firmware driver API description

33.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implementing the HAL_I2C_MspInit() API:
 - a. Enable the I2Cx interface clock
 - b. I2C pins configuration
 - Enable the clock for the I2C GPIOs
 - Configure I2C pins as alternate function open-drain
 - c. NVIC configuration if you need to use interrupt process
 - Configure the I2Cx interrupt priority
 - Enable the NVIC I2C IRQ Channel
 - d. DMA Configuration if you need to use DMA process
 - Declare a DMA_HandleTypeDef handle structure for the transmit or receive stream
 - Enable the DMAx interface clock using
 - Configure the DMA handle parameters
 - Configure the DMA Tx or Rx stream
 - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx stream
3. Configure the Communication Clock Timing, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()

- Transmit in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO sequential operation

Note:

These interfaces allow to manage a sequential transfer with a repeated start condition when a direction change during transfer

- A specific option field manage the different steps of a sequential transfer
- Option field values are defined through @ref I2C_XFEROPTIONS and are listed below:
 - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functionnal is same as associated interfaces in no sequential mode
 - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
 - I2C_FIRST_AND_NEXT_FRAME: Sequential usage (Master only), this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition, an then permit a call the same master sequential interface several times (like HAL_I2C_Master_Sequential_Transmit_IT() then HAL_I2C_Master_Sequential_Transmit_IT())
 - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
 - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
- Differents sequential I2C interfaces are listed below:
 - Sequential transmit in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Sequential_Transmit_IT()
 - At transmission end of current frame transfer, HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
 - Sequential receive in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Sequential_Receive_IT()
 - At reception end of current frame transfer, HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
 - Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
 - End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
 - Enable/disable the Address listen mode in slave I2C mode using HAL_I2C_EnableListen_IT() HAL_I2C_DisableListen_IT()
 - When address slave I2C match, HAL_I2C_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master (Write/Read).
 - At Listen mode end HAL_I2C_ListenCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_ListenCpltCallback()
 - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Sequential_Transmit_IT()

- At transmission end of current frame transfer, HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Sequential receive in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Sequential_Receive_IT()
 - At reception end of current frame transfer, HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO MEM operation

- Write an amount of data in non-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()

DMA mode IO operation

- Transmit in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

DMA mode IO MEM operation

- Write an amount of data in non-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()

- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral
- __HAL_I2C_GENERATE_NACK: Generate a Non-Acknowledge I2C peripheral in Slave mode
- __HAL_I2C_GET_FLAG: Check whether the specified I2C flag is set or not
- __HAL_I2C_CLEAR_FLAG: Clear the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt

Note: You can refer to the I2C HAL driver header file for more useful macros

33.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
 - Clock Timing
 - Own Address 1
 - Addressing mode (Master, Slave)
 - Dual Addressing mode
 - Own Address 2
 - Own Address 2 Mask
 - General call mode
 - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- [HAL_I2C_Init](#)
- [HAL_I2C_DeInit](#)
- [HAL_I2C_MspInit](#)
- [HAL_I2C_MspDeInit](#)

33.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:
 - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :

- HAL_I2C_Master_Transmit()
 - HAL_I2C_Master_Receive()
 - HAL_I2C_Slave_Transmit()
 - HAL_I2C_Slave_Receive()
 - HAL_I2C_Mem_Write()
 - HAL_I2C_Mem_Read()
 - HAL_I2C_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
- HAL_I2C_Master_Transmit_IT()
 - HAL_I2C_Master_Receive_IT()
 - HAL_I2C_Slave_Transmit_IT()
 - HAL_I2C_Slave_Receive_IT()
 - HAL_I2C_Mem_Write_IT()
 - HAL_I2C_Mem_Read_IT()
4. No-Blocking mode functions with DMA are :
- HAL_I2C_Master_Transmit_DMA()
 - HAL_I2C_Master_Receive_DMA()
 - HAL_I2C_Slave_Transmit_DMA()
 - HAL_I2C_Slave_Receive_DMA()
 - HAL_I2C_Mem_Write_DMA()
 - HAL_I2C_Mem_Read_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
- HAL_I2C_MemTxCpltCallback()
 - HAL_I2C_MemRxCpltCallback()
 - HAL_I2C_MasterTxCpltCallback()
 - HAL_I2C_MasterRxCpltCallback()
 - HAL_I2C_SlaveTxCpltCallback()
 - HAL_I2C_SlaveRxCpltCallback()
 - HAL_I2C_ErrorCallback()

This section contains the following APIs:

- [**HAL_I2C_Master_Transmit**](#)
- [**HAL_I2C_Master_Receive**](#)
- [**HAL_I2C_Slave_Transmit**](#)
- [**HAL_I2C_Slave_Receive**](#)
- [**HAL_I2C_Master_Transmit_IT**](#)
- [**HAL_I2C_Master_Receive_IT**](#)
- [**HAL_I2C_Slave_Transmit_IT**](#)
- [**HAL_I2C_Slave_Receive_IT**](#)
- [**HAL_I2C_Master_Transmit_DMA**](#)
- [**HAL_I2C_Master_Receive_DMA**](#)
- [**HAL_I2C_Slave_Transmit_DMA**](#)
- [**HAL_I2C_Slave_Receive_DMA**](#)
- [**HAL_I2C_Mem_Write**](#)
- [**HAL_I2C_Mem_Read**](#)
- [**HAL_I2C_Mem_Write_IT**](#)
- [**HAL_I2C_Mem_Read_IT**](#)
- [**HAL_I2C_Mem_Write_DMA**](#)
- [**HAL_I2C_Mem_Read_DMA**](#)
- [**HAL_I2C_IsDeviceReady**](#)

- [*HAL_I2C_Master_SequENTIAL_Transmit_IT*](#)
- [*HAL_I2C_Master_SequENTIAL_Receive_IT*](#)
- [*HAL_I2C_Slave_SequENTIAL_Transmit_IT*](#)
- [*HAL_I2C_Slave_SequENTIAL_Receive_IT*](#)
- [*HAL_I2C_EnableListen_IT*](#)
- [*HAL_I2C_DisableListen_IT*](#)
- [*HAL_I2C_Master_Abort_IT*](#)

33.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [*HAL_I2C_GetState*](#)
- [*HAL_I2C_GetMode*](#)
- [*HAL_I2C_GetError*](#)

33.2.5 Detailed description of functions

HAL_I2C_Init

Function name

HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)

Function description

Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and initialize the associated handle.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DeInit

Function name

HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C peripheral.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_MspInit

Function name

void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)

Function description

Initialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MspInit

Function name

void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_Master_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Master_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit

Function name

```
HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Transmits in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Receive

Function name

```
HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Receive in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Write

Function name

```
HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t  
MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)
```

Function description

Write an amount of data in blocking mode to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Read

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Read an amount of data in blocking mode from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Checks if target device is ready for communication.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used with Memory devices

HAL_I2C_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with Interrupt to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Read an amount of data in non-blocking mode with Interrupt from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address

- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_SequENTIAL_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Sequential_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of
 - I2C_XferOptions_definition

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_SequENTIAL_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Sequential_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of
 - I2C_XferOptions_definition

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Sequential_Transmit_IT

Function name

```
HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData,  
uint16_t Size, uint32_t XferOptions)
```

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of
 - I2C_XferOptions_definition

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Sequential_Receive_IT

Function name

```
HAL_StatusTypeDef HAL_I2C_Slave_Sequential_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData,  
uint16_t Size, uint32_t XferOptions)
```

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of
 - I2C_XferOptions_definition

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EnableListen_IT

Function name

```
HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)
```

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C

Return values

- **HAL:** status

HAL_I2C_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)

Function description

Abort a master/host I2C process communication with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address

Return values

- **HAL:** status

HAL_I2C_Master_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_DMA

Function name

```
HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress,  
uint8_t * pData, uint16_t Size)
```

Function description

Receive in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_DMA

Function name

```
HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t t  
Size)
```

Function description

Transmit in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_DMA

Function name

```
HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t t  
Size)
```

Function description

Receive in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_DMA

Function name

```
HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress,  
uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
```

Function description

Write an amount of data in non-blocking mode with DMA to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_DMA

Function name

```
HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress,  
uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
```

Function description

Reads an amount of data in non-blocking mode with DMA from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be read

Return values

- **HAL:** status

HAL_I2C_EV_IRQHandler

Function name

```
void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)
```

Function description

This function handles I2C event interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ER IRQHandler

Function name

void HAL_I2C_ER IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C error interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterTxCpltCallback

Function name

void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterRxCpltCallback

Function name

void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveTxCpltCallback

Function name

void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveRxCpltCallback

Function name

void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AddrCallback

Function name

void HAL_I2C_AddrCallback (I2C_HandleTypeDef * hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **TransferDirection:** Master request Transfer Direction (Write/Read), value of
 - I2C_XferOptions_definition
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_I2C_ListenCpltCallback

Function name

void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Listen Complete callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemTxCpltCallback

Function name

```
void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)
```

Function description

Memory Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemRxCpltCallback

Function name

```
void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)
```

Function description

Memory Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ErrorCallback

Function name

```
void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)
```

Function description

I2C error callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AbortCpltCallback

Function name

```
void HAL_I2C_AbortCpltCallback (I2C_HandleTypeDef * hi2c)
```

Function description

I2C abort callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_GetState**Function name****HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)****Function description**

Return the I2C handle state.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** state

HAL_I2C_GetMode**Function name****HAL_I2C_ModeTypeDef HAL_I2C_GetMode (I2C_HandleTypeDef * hi2c)****Function description**

Returns the I2C Master, Slave, Memory or no mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

Return values

- **HAL:** mode

HAL_I2C_GetError**Function name****uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)****Function description**

Return the I2C error code.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **I2C:** Error Code

33.3 I2C Firmware driver defines

33.3.1 I2C

*I2C Addressing Mode***I2C_ADDRESSINGMODE_7BIT****I2C_ADDRESSINGMODE_10BIT**

I2C Dual Addressing Mode**I2C_DUALADDRESS_DISABLE****I2C_DUALADDRESS_ENABLE****I2C Error Code definition****HAL_I2C_ERROR_NONE**

No error

HAL_I2C_ERROR_BERR

BERR error

HAL_I2C_ERROR_ARLO

ARLO error

HAL_I2C_ERROR_AF

ACKF error

HAL_I2C_ERROR_OVR

OVR error

HAL_I2C_ERROR_DMA

DMA transfer error

HAL_I2C_ERROR_TIMEOUT

Timeout error

HAL_I2C_ERROR_SIZE

Size Management error

I2C Exported Macros**_HAL_I2C_RESET_HANDLE_STATE****Description:**

- Reset I2C handle state.

Parameters:

- **_HANDLE_**: specifies the I2C Handle.

Return value:

- None

_HAL_I2C_ENABLE_IT**Description:**

- Enable the specified I2C interrupt.

Parameters:

- **_HANDLE_**: specifies the I2C Handle.
- **_INTERRUPT_**: specifies the interrupt source to enable. This parameter can be one of the following values:
 - I2C_IT_ERRI Errors interrupt enable
 - I2C_IT_TCI Transfer complete interrupt enable
 - I2C_IT_STOPI STOP detection interrupt enable
 - I2C_IT_NACKI NACK received interrupt enable

- I2C_IT_ADDRI Address match interrupt enable
- I2C_IT_RXI RX interrupt enable
- I2C_IT_TXI TX interrupt enable

Return value:

- None

[__HAL_I2C_DISABLE_IT](#)**Description:**

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
 - I2C_IT_ERRI Errors interrupt enable
 - I2C_IT_TCI Transfer complete interrupt enable
 - I2C_IT_STOPI STOP detection interrupt enable
 - I2C_IT_NACKI NACK received interrupt enable
 - I2C_IT_ADDRI Address match interrupt enable
 - I2C_IT_RXI RX interrupt enable
 - I2C_IT_TXI TX interrupt enable

Return value:

- None

[__HAL_I2C_GET_IT_SOURCE](#)**Description:**

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
 - I2C_IT_ERRI Errors interrupt enable
 - I2C_IT_TCI Transfer complete interrupt enable
 - I2C_IT_STOPI STOP detection interrupt enable
 - I2C_IT_NACKI NACK received interrupt enable
 - I2C_IT_ADDRI Address match interrupt enable
 - I2C_IT_RXI RX interrupt enable
 - I2C_IT_TXI TX interrupt enable

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

[__HAL_I2C_GET_FLAG](#)**Description:**

- Check whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - I2C_FLAG_TXE Transmit data register empty

- I2C_FLAG_TXIS Transmit interrupt status
- I2C_FLAG_RXNE Receive data register not empty
- I2C_FLAG_ADDR Address matched (slave mode)
- I2C_FLAG_AF Acknowledge failure received flag
- I2C_FLAG_STOPF STOP detection flag
- I2C_FLAG_TC Transfer complete (master mode)
- I2C_FLAG_TCR Transfer complete reload
- I2C_FLAG_BERR Bus error
- I2C_FLAG_ARLO Arbitration lost
- I2C_FLAG_OVR Overrun/Underrun
- I2C_FLAG_PECERR PEC error in reception
- I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
- I2C_FLAG_ALERT SMBus alert
- I2C_FLAG_BUSY Bus busy
- I2C_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

[__HAL_I2C_CLEAR_FLAG](#)**Description:**

- Clear the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
 - I2C_FLAG_TXE Transmit data register empty
 - I2C_FLAG_ADDR Address matched (slave mode)
 - I2C_FLAG_AF Acknowledge failure received flag
 - I2C_FLAG_STOPF STOP detection flag
 - I2C_FLAG_BERR Bus error
 - I2C_FLAG_ARLO Arbitration lost
 - I2C_FLAG_OVR Overrun/Underrun
 - I2C_FLAG_PECERR PEC error in reception
 - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
 - I2C_FLAG_ALERT SMBus alert

Return value:

- None

[__HAL_I2C_ENABLE](#)**Description:**

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

[__HAL_I2C_DISABLE](#)**Description:**

- Disable the specified I2C peripheral.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.

Return value:

- None

`__HAL_I2C_GENERATE_NACK`**Description:**

- Generate a Non-Acknowledge I2C peripheral in Slave mode.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.

Return value:

- None

I2C Flag definition**`I2C_FLAG_TXE`****`I2C_FLAG_TXIS`****`I2C_FLAG_RXNE`****`I2C_FLAG_ADDR`****`I2C_FLAG_AF`****`I2C_FLAG_STOPF`****`I2C_FLAG_TC`****`I2C_FLAG_TCR`****`I2C_FLAG_BERR`****`I2C_FLAG_ARLO`****`I2C_FLAG_OVR`****`I2C_FLAG_PECERR`****`I2C_FLAG_TIMEOUT`****`I2C_FLAG_ALERT`****`I2C_FLAG_BUSY`****`I2C_FLAG_DIR`*****I2C General Call Addressing Mode*****`I2C_GENERALCALL_DISABLE`****`I2C_GENERALCALL_ENABLE`**

I2C Interrupt configuration definition`I2C_IT_ERRI``I2C_IT_TCI``I2C_IT_STOPI``I2C_IT_NACKI``I2C_IT_ADDRI``I2C_IT_RXI``I2C_IT_TXI`*I2C Memory Address Size*`I2C_MEMADD_SIZE_8BIT``I2C_MEMADD_SIZE_16BIT`*I2C No-Stretch Mode*`I2C_NOSTRETCH_DISABLE``I2C_NOSTRETCH_ENABLE`*I2C Own Address2 Masks*`I2C_OA2_NOMASK``I2C_OA2_MASK01``I2C_OA2_MASK02``I2C_OA2_MASK03``I2C_OA2_MASK04``I2C_OA2_MASK05``I2C_OA2_MASK06``I2C_OA2_MASK07`*I2C Reload End Mode*`I2C_RELOAD_MODE``I2C_AUTOEND_MODE``I2C_SOFTEND_MODE`*I2C Start or Stop Mode*`I2C_NO_STARTSTOP``I2C_GENERATE_STOP`

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

I2C Transfer Direction

I2C_DIRECTION_TRANSMIT

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_NO_OPTION_FRAME

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

34 HAL I2C Extension Driver

34.1 I2CEEx Firmware driver API description

34.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32H7XX devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop modes
- Disable or enable Fast Mode Plus

34.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function `HAL_I2CEEx_ConfigAnalogFilter()`
2. Configure I2C Digital noise filter using the function `HAL_I2CEEx_ConfigDigitalFilter()`
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
 - `HAL_I2CEEx_EnableWakeUp()`
 - `HAL_I2CEEx_DisableWakeUp()`
4. Configure the enable or disable of fast mode plus driving capability using the functions :
 - `HAL_I2CEEx_EnableFastModePlus()`
 - `HAL_I2CEEx_DisableFastModePlus()`

34.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters
- Configure Wake Up Feature
- Configure Fast Mode Plus

This section contains the following APIs:

- [`HAL_I2CEEx_ConfigAnalogFilter`](#)
- [`HAL_I2CEEx_ConfigDigitalFilter`](#)
- [`HAL_I2CEEx_EnableWakeUp`](#)
- [`HAL_I2CEEx_DisableWakeUp`](#)
- [`HAL_I2CEEx_EnableFastModePlus`](#)
- [`HAL_I2CEEx_DisableFastModePlus`](#)

34.1.4 Detailed description of functions

`HAL_I2CEEx_ConfigAnalogFilter`

Function name

`HAL_StatusTypeDef HAL_I2CEEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)`

Function description

Configure I2C Analog noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **AnalogFilter:** New state of the Analog filter.

Return values

- **HAL:** status

HAL_I2CEx_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)

Function description

Configure I2C Digital noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **DigitalFilter:** Coefficient of digital noise filter between 0x00 and 0x0F.

Return values

- **HAL:** status

HAL_I2CEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Enable I2C wakeup from stop mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Disable I2C wakeup from stop mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_EnableFastModePlus**Function name**

```
void HAL_I2CEx_EnableFastModePlus (uint32_t ConfigFastModePlus)
```

Function description

Enable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values

Return values

- **None:**

HAL_I2CEx_DisableFastModePlus**Function name**

```
void HAL_I2CEx_DisableFastModePlus (uint32_t ConfigFastModePlus)
```

Function description

Disable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values

Return values

- **None:**

34.2 I2CEx Firmware driver defines

34.2.1 I2CEx

*I2C Extended Analog Filter***I2C_ANALOGFILTER_ENABLE****I2C_ANALOGFILTER_DISABLE***I2C Extended Fast Mode Plus***I2C_FASTMODEPLUS_PB6****I2C_FASTMODEPLUS_PB7****I2C_FASTMODEPLUS_PB8****I2C_FASTMODEPLUS_PB9****I2C_FASTMODEPLUS_I2C1****I2C_FASTMODEPLUS_I2C2****I2C_FASTMODEPLUS_I2C3**

I2C_FASTMODEPLUS_I2C4

35 HAL I2S Generic Driver

35.1 I2S Firmware driver registers structures

35.1.1 I2S_InitTypeDef

Data Fields

- `uint32_t Mode`
- `uint32_t Standard`
- `uint32_t DataFormat`
- `uint32_t MCLKOutput`
- `uint32_t AudioFreq`
- `uint32_t CPOL`
- `uint32_t FirstBit`
- `uint32_t WSInversion`
- `uint32_t IOSwap`
- `uint32_t Data24BitAlignment`
- `uint32_t FifoThreshold`
- `uint32_t MasterKeepIOState`
- `uint32_t SlaveExtendFREDetection`

Field Documentation

- `uint32_t I2S_InitTypeDef::Mode`

Specifies the I2S operating mode. This parameter can be a value of **I2S Mode**

- `uint32_t I2S_InitTypeDef::Standard`

Specifies the standard used for the I2S communication. This parameter can be a value of **I2S Standard**

- `uint32_t I2S_InitTypeDef::DataFormat`

Specifies the data format for the I2S communication. This parameter can be a value of **I2S Data Format**

- `uint32_t I2S_InitTypeDef::MCLKOutput`

Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of **I2S MCLK Output**

- `uint32_t I2S_InitTypeDef::AudioFreq`

Specifies the frequency selected for the I2S communication. This parameter can be a value of **I2S Audio Frequency**

- `uint32_t I2S_InitTypeDef::CPOL`

Specifies the idle state of the I2S clock. This parameter can be a value of **I2S Clock Polarity**

- `uint32_t I2S_InitTypeDef::FirstBit`

Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of **I2S MSB LSB Transmission**

- `uint32_t I2S_InitTypeDef::WSInversion`

Control the Word Select Inversion. This parameter can be a value of **I2S Word Select Inversion**

- `uint32_t I2S_InitTypeDef::IOSwap`

Invert MISO/MOSI alternate functions This parameter can be a value of **Control I2S IO Swap**

- **`uint32_t I2S_InitTypeDef::Data24BitAlignment`**
Specifies the Data Padding for 24 bits data lenght This parameter can be a value of **`Data Padding 24Bit`**
- **`uint32_t I2S_InitTypeDef::FifoThreshold`**
Specifies the FIFO threshold level. This parameter can be a value of **`I2S Fifo Threshold`**
- **`uint32_t I2S_InitTypeDef::MasterKeepIOState`**
Control of Alternate function GPIOs state This parameter can be a value of **`Keep IO State`**
- **`uint32_t I2S_InitTypeDef::SlaveExtendFREDetection`**
Control the channel length in SLAVE. This parameter can be a value of **`Slave Extend FRE Detection`**

35.1.2 `__I2S_HandleTypeDef`

Data Fields

- **`SPI_TypeDef * Instance`**
- **`I2S_InitTypeDef Init`**
- **`uint16_t * pTxBuffPtr`**
- **`__IO uint16_t TxXferSize`**
- **`__IO uint16_t TxXferCount`**
- **`uint16_t * pRxBuffPtr`**
- **`__IO uint16_t RxXferSize`**
- **`__IO uint16_t RxXferCount`**
- **`void(* RxISR`**
- **`void(* TxISR`**
- **`DMA_HandleTypeDef * hdmatx`**
- **`DMA_HandleTypeDef * hdmarx`**
- **`__IO HAL_LockTypeDef Lock`**
- **`__IO HAL_I2S_StateTypeDef State`**
- **`__IO uint32_t ErrorCode`**

Field Documentation

- **`SPI_TypeDef* __I2S_HandleTypeDef::Instance`**
I2S registers base address
- **`I2S_InitTypeDef __I2S_HandleTypeDef::Init`**
I2S communication parameters
- **`uint16_t* __I2S_HandleTypeDef::pTxBuffPtr`**
Pointer to I2S Tx transfer buffer
- **`__IO uint16_t __I2S_HandleTypeDef::TxXferSize`**
I2S Tx transfer size
- **`__IO uint16_t __I2S_HandleTypeDef::TxXferCount`**
I2S Tx transfer Counter
- **`uint16_t* __I2S_HandleTypeDef::pRxBuffPtr`**
Pointer to I2S Rx transfer buffer
- **`__IO uint16_t __I2S_HandleTypeDef::RxXferSize`**
I2S Rx transfer size
- **`__IO uint16_t __I2S_HandleTypeDef::RxXferCount`**
I2S Rx transfer counter

- **void(* __I2S_HandleTypeDef::RxISR)(struct __I2S_HandleTypeDef *hi2s)**
function pointer on Rx ISR
- **void(* __I2S_HandleTypeDef::TxISR)(struct __I2S_HandleTypeDef *hi2s)**
function pointer on Tx ISR
- **DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmatx**
I2S Tx DMA handle parameters
- **DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmarx**
I2S Rx DMA handle parameters
- **_IO HAL_LockTypeDef __I2S_HandleTypeDef::Lock**
I2S locking object
- **_IO HAL_I2S_StateTypeDef __I2S_HandleTypeDef::State**
I2S communication state
- **_IO uint32_t __I2S_HandleTypeDef::ErrorCode**
I2S Error code

35.2 I2S Firmware driver API description

35.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
 - a. Enable the SPIx interface clock.
 - b. I2S pins configuration:
 - Enable the clock for the I2S GPIOs.
 - Configure these I2S pins as alternate function pull-up.
 - c. NVIC configuration if you need to use interrupt process (HAL_I2S_Transmit_IT() and HAL_I2S_Receive_IT() APIs).
 - Configure the I2Sx interrupt priority.
 - Enable the NVIC I2S IRQ handle.
 - d. DMA Configuration if you need to use DMA process (HAL_I2S_Transmit_DMA() and HAL_I2S_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx Channel.
 - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Channel.
3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_I2S_Init() function.

Note: The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_I2S_ENABLE_IT() and __HAL_I2S_DISABLE_IT() inside the transmit and receive process.

Note: Make sure that either:

- External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the *stm32h7xx_hal_conf.h* file. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()
- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

Below the list of most used macros in I2S HAL driver.

- __HAL_I2S_ENABLE: Enable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_DISABLE: Disable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_ENABLE_IT : Enable the specified I2S interrupts
- __HAL_I2S_DISABLE_IT : Disable the specified I2S interrupts
- __HAL_I2S_GET_FLAG: Check whether the specified I2S flag is set or not

Note:

You can refer to the I2S HAL driver header file for more useful macros

35.2.2 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
 - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.

- No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
 - HAL_I2S_Transmit()
 - HAL_I2S_Receive()
 3. No-Blocking mode functions with Interrupt are :
 - HAL_I2S_Transmit_IT()
 - HAL_I2S_Receive_IT()
 4. No-Blocking mode functions with DMA are :
 - HAL_I2S_Transmit_DMA()
 - HAL_I2S_Receive_DMA()
 5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_I2S_TxCpltCallback()
 - HAL_I2S_TxHalfCpltCallback()
 - HAL_I2S_RxCpltCallback()
 - HAL_I2S_RxHalfCpltCallback()
 - HAL_I2S_ErrorCallback()

This section contains the following APIs:

- [**HAL_I2S_Transmit**](#)
- [**HAL_I2S_Receive**](#)
- [**HAL_I2S_Transmit_IT**](#)
- [**HAL_I2S_Receive_IT**](#)
- [**HAL_I2S_Transmit_DMA**](#)
- [**HAL_I2S_Receive_DMA**](#)
- [**HAL_I2S_DMAPause**](#)
- [**HAL_I2S_DMAResume**](#)
- [**HAL_I2S_DMAStop**](#)
- [**HAL_I2S_IRQHandler**](#)
- [**HAL_I2S_TxHalfCpltCallback**](#)
- [**HAL_I2S_TxCpltCallback**](#)
- [**HAL_I2S_RxHalfCpltCallback**](#)
- [**HAL_I2S_RxCpltCallback**](#)
- [**HAL_I2S_ErrorCallback**](#)

35.2.3

Peripheral State and Errors functions

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

This section contains the following APIs:

- [**HAL_I2S_GetState**](#)
- [**HAL_I2S_GetError**](#)

35.2.4

Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze the I2Sx peripheral in simplex mode:

- User must Implement HAL_I2S_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2S_Init() to configure the selected device with the selected configuration:
 - Mode

- Standard
 - Data Format
 - MCLK Output
 - Audio frequency
 - Polarity
 - First Bit
 - WS Inversion
 - IO Swap
 - Data 24Bit Alignment
 - Fifo Threshold
 - Alternate function GPIOs state
 - Channel length in SLAVE
- Call the function HAL_I2S_DelInit() to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- [**HAL_I2S_Init**](#)
- [**HAL_I2S_DelInit**](#)
- [**HAL_I2S_MspInit**](#)
- [**HAL_I2S_MspDelInit**](#)

35.2.5 Detailed description of functions

HAL_I2S_Init

Function name

HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)

Function description

Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DelInit

Function name

HAL_StatusTypeDef HAL_I2S_DelInit (I2S_HandleTypeDef * hi2s)

Function description

DeInitializes the I2S peripheral.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_MspInit

Function name

void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP Init.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_MspInit

Function name

```
void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)
```

Function description

I2S MSP Delinit.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_Transmit

Function name

```
HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of frames to be sent.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

HAL_I2S_Receive

Function name

```
HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Receive an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of frames to be sent.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction.
- This function can use an Audio Frequency up to 44KHz when I2S Clock Source is 32MHz

HAL_I2S_Transmit_IT

Function name

`HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)`

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

HAL_I2S_Receive_IT

Function name

`HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)`

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- It is recommended to use DMA for the I2S receiver to avoid de-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.
- This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

HAL_I2S_IRQHandler

Function name

```
void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)
```

Function description

This function handles I2S interrupt request.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_Transmit_DMA

Function name

```
HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
```

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Transmit data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_DMA

Function name

```
HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
```

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_DMAPause

Function name

HAL_StatusTypeDef HAL_I2S_DM_PAUSE (I2S_HandleTypeDef * hi2s)

Function description

Pauses the audio stream playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAResume

Function name

HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)

Function description

Resumes the audio stream playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAStop

Function name

HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)

Function description

Stops the audio stream playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_TxHalfCpltCallback

Function name

```
void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
```

Function description

Tx Transfer Half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_TxCpltCallback

Function name

```
void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)
```

Function description

Tx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxHalfCpltCallback

Function name

```
void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
```

Function description

Rx Transfer half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxCpltCallback

Function name

```
void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)
```

Function description

Rx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_ErrorCallback

Function name

```
void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)
```

Function description

I2S error callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_GetState

Function name

```
HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)
```

Function description

Return the I2S state.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** state

HAL_I2S_GetError

Function name

```
uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)
```

Function description

Return the I2S error code.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **I2S:** Error Code

35.3 I2S Firmware driver defines

35.3.1 I2S

I2S Audio Frequency

I2S_AUDIOFREQ_192K

I2S_AUDIOFREQ_96K

I2S_AUDIOFREQ_48K

I2S_AUDIOFREQ_44K

I2S_AUDIOFREQ_32K

I2S_AUDIOFREQ_22K

I2S_AUDIOFREQ_16K

I2S_AUDIOFREQ_11K

I2S_AUDIOFREQ_8K

I2S_AUDIOFREQ_DEFAULT

I2S Clock Polarity

I2S_CPOL_LOW

I2S_CPOL_HIGH

Data Padding 24Bit

I2S_DATA_24BIT_ALIGNMENT_RIGHT

I2S_DATA_24BIT_ALIGNMENT_LEFT

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error Defintion

HAL_I2S_ERROR_NONE

No error

HAL_I2S_ERROR_UDR

I2S Underrun error

HAL_I2S_ERROR_OVR

I2S Overrun error

HAL_I2S_ERROR_FRE

I2S Frame format error

HAL_I2S_ERROR_DMA

DMA transfer error

HAL_I2S_ERROR_DMA

DMA transfer error

HAL_I2S_ERROR_TIMEOUT

Timeout error

HAL_I2S_ERROR_PRESCALER

Prescaler error

I2S Exported Macros

[__HAL_I2S_RESET_HANDLE_STATE](#)

Description:

- Reset I2S handle state.

Parameters:

- [__HANDLE__](#): specifies the I2S Handle.

Return value:

- None

[__HAL_I2S_ENABLE](#)

Description:

- Enable the specified SPI peripheral (in I2S mode).

Parameters:

- [__HANDLE__](#): specifies the I2S Handle.

Return value:

- None

[__HAL_I2S_DISABLE](#)

Description:

- Disable the specified SPI peripheral (in I2S mode).

Parameters:

- [__HANDLE__](#): specifies the I2S Handle.

Return value:

- None

[__HAL_I2S_ENABLE_IT](#)

Description:

- Enable the specified I2S interrupts.

Parameters:

- [__HANDLE__](#): specifies the I2S Handle.
- [__INTERRUPT__](#): specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - I2S_IT_TXE: Tx buffer empty interrupt enable
 - I2S_IT_RXNE: RX buffer not empty interrupt enable
 - I2S_IT_ERR: Error interrupt enable

Return value:

- None

[__HAL_I2S_DISABLE_IT](#)

Description:

- Disable the specified I2S interrupts.

Parameters:

- [__HANDLE__](#): specifies the I2S Handle.
- [__INTERRUPT__](#): specifies the interrupt source to enable or disable. This parameter can be one of the following values:

- I2S_IT_TXE: Tx buffer empty interrupt enable
- I2S_IT_RXNE: RX buffer not empty interrupt enable
- I2S_IT_ERR: Error interrupt enable

Return value:

- None

_HAL_I2S_GET_IT_SOURCE

Description:

- Checks if the specified I2S interrupt source is enabled or disabled.

Parameters:

- HANDLE: specifies the I2S Handle. This parameter can be I2S where x: 1, 2, or 3 to select the I2S peripheral.
- INTERRUPT: specifies the I2S interrupt source to check. This parameter can be one of the following values:
 - I2S_IT_TXE: Tx buffer empty interrupt enable
 - I2S_IT_RXNE: RX buffer not empty interrupt enable
 - I2S_IT_ERR: Error interrupt enable

Return value:

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

_HAL_I2S_GET_FLAG

Description:

- Checks whether the specified I2S flag is set or not.

Parameters:

- HANDLE: specifies the I2S Handle.
- FLAG: specifies the flag to check. This parameter can be one of the following values:
 - I2S_FLAG_TXE : Tx buffer empty flag
 - I2S_FLAG_RXNE : Rx buffer not empty flag
 - I2S_FLAG_UDR : Underrun flag
 - I2S_FLAG_OVR : Overrun flag
 - I2S_FLAG_FRE : TI mode frame format error flag

Return value:

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

_HAL_I2S_CLEAR_UDRFLAG

Description:

- Clears the I2S UDR pending flag.

Parameters:

- HANDLE: specifies the I2S Handle.

Return value:

- None

_HAL_I2S_CLEAR_OVRFIELD

Description:

- Clears the I2S OVR pending flag.

Parameters:

- HANDLE: specifies the I2S Handle.

Return value:

- None

_HAL_I2S_CLEAR_FREFLAG**Description:**

- Clear the I2S FRE pending flag.

Parameters:

- `_HANDLE_`: specifies the I2S Handle.

Return value:

- None

*I2S Fifo Threshold***I2S_FIFO_THRESHOLD_01DATA****I2S_FIFO_THRESHOLD_02DATA****I2S_FIFO_THRESHOLD_03DATA****I2S_FIFO_THRESHOLD_04DATA****I2S_FIFO_THRESHOLD_05DATA****I2S_FIFO_THRESHOLD_06DATA****I2S_FIFO_THRESHOLD_07DATA****I2S_FIFO_THRESHOLD_08DATA***I2S Flag definition***I2S_FLAG_TXE****I2S_FLAG_RXNE****I2S_FLAG_UDR****I2S_FLAG_RXWNE****I2S_FLAG_OVR****I2S_FLAG_FRE***I2S Interrupt definition***I2S_IT_TXE****I2S_IT_RXNE****I2S_IT_ERR***Control I2S IO Swap***I2S_IO_SWAP_DISABLE****I2S_IO_SWAP_ENABLE**

Keep IO State`I2S_MASTER_KEEP_IO_STATE_DISABLE``I2S_MASTER_KEEP_IO_STATE_ENABLE`***I2S MCLK Output***`I2S_MCLKOUTPUT_ENABLE``I2S_MCLKOUTPUT_DISABLE`***I2S Mode***`I2S_MODE_SLAVE_TX``I2S_MODE_SLAVE_RX``I2S_MODE_MASTER_TX``I2S_MODE_MASTER_RX``I2S_MODE_SLAVE_FD``I2S_MODE_MASTER_FD`***I2S MSB LSB Transmission***`I2S_FIRSTBIT_MSB``I2S_FIRSTBIT_LSB`***Slave Extend FRE Detection***`I2S_SLAVE_EXTEND_FRE_DETECTION_DISABLE``I2S_SLAVE_EXTEND_FRE_DETECTION_ENABLE`***I2S Standard***`I2S_STANDARD_PHILIPS``I2S_STANDARD_MSB``I2S_STANDARD_LSB``I2S_STANDARD_PCM_SHORT``I2S_STANDARD_PCM_LONG`***I2S Word Select Inversion***`I2S_WS_INVERSION_DISABLE``I2S_WS_INVERSION_ENABLE`

36 HAL I2S Extension Driver

36.1 I2SEEx Firmware driver API description

36.1.1 I2S Extension features

- In I2S full duplex mode, SPI2S peripheral is able to manage sending and receiving data simultaneously using two data lines.

36.1.2 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
 - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
 - HAL_I2SEEx_TransmitReceive()
3. No-Blocking mode functions with Interrupt are :
 - HAL_I2SEEx_TransmitReceive_IT()
4. No-Blocking mode functions with DMA are :
 - HAL_I2SEEx_TransmitReceive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_I2SEEx_TxRxCpltCallback()
 - HAL_I2SEEx_TxRxErrorCallback()

This section contains the following APIs:

- [**HAL_I2SEEx_TransmitReceive**](#)
- [**HAL_I2SEEx_TransmitReceive_IT**](#)
- [**HAL_I2SEEx_TransmitReceive_DMA**](#)
- [**HAL_I2SEEx_TxRxHalfCpltCallback**](#)
- [**HAL_I2SEEx_TxRxCpltCallback**](#)

36.1.3 Detailed description of functions

HAL_I2SEEx_TransmitReceive

Function name

`HAL_StatusTypeDef HAL_I2SEEx_TransmitReceive (I2S_HandleTypeDef * hi2s, uint16_t * pTxData, uint16_t * pRxData, uint16_t Size, uint32_t Timeout)`

Function description

Transmit and Receive an amount of data in blocking mode.

Parameters

- **hi2s**: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData**: a 16-bit pointer to the Transmit data buffer
- **pRxData**: a 16-bit pointer to the Receive data buffer
- **Size**: number of frames to be sent

- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

HAL_I2SEx_TransmitReceive_IT

Function name

```
HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pTxData,  
uint16_t * pRxData, uint16_t Size)
```

Function description

Transmit and Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData:** a 16-bit pointer to the Transmit data buffer.
- **pRxData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- This function can use an Audio Frequency up to 48KHz when I2S Clock Source is 32MHz

HAL_I2SEx_TransmitReceive_DMA

Function name

```
HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pTxData,  
uint16_t * pRxData, uint16_t Size)
```

Function description

Transmit and Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pTxData:** a 16-bit pointer to the Transmit data buffer.
- **pRxData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of frames to be sent.

Return values

- **HAL:** status

Notes

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2SEEx_TxRxHalfCpltCallback

Function name

void HAL_I2SEEx_TxRxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx/Rx Transfer half completed callbacks.

Parameters

- hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- None:**

HAL_I2SEEx_TxRxCpltCallback

Function name

void HAL_I2SEEx_TxRxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx/Rx Transfer completed callbacks.

Parameters

- hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- None:**

37 HAL IRDA Generic Driver

37.1 IRDA Firmware driver registers structures

37.1.1 IRDA_InitTypeDef

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t Parity*
- *uint32_t Mode*
- *uint8_t Prescaler*
- *uint16_t PowerMode*

Field Documentation

- *uint32_t IRDA_InitTypeDef::BaudRate*

This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hirda->Init.BaudRate)))

- *uint32_t IRDA_InitTypeDef::WordLength*

Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of **IRDA Word Length**

- *uint32_t IRDA_InitTypeDef::Parity*

Specifies the parity mode. This parameter can be a value of **IRDA Parity**

Note:

- When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- *uint32_t IRDA_InitTypeDef::Mode*

Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of **IRDA Transfer Mode**

- *uint8_t IRDA_InitTypeDef::Prescaler*

Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.

Note:

- Prescaler value 0 is forbidden
- *uint16_t IRDA_InitTypeDef::PowerMode*

Specifies the IRDA power mode. This parameter can be a value of **IRDA Low Power**

37.1.2 IRDA_HandleTypeDef

Data Fields

- *USART_TypeDef * Instance*
- *IRDA_InitTypeDef Init*
- *uint8_t * pTxBuffPtr*
- *uint16_t TxXferSize*
- *_IO uint16_t TxXferCount*
- *uint8_t * pRxBuffPtr*

- `uint16_t RxXferSize`
- `__IO uint16_t RxXferCount`
- `uint16_t Mask`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `__IO HAL_IRDA_StateTypeDef gState`
- `__IO HAL_IRDA_StateTypeDef RxState`
- `uint32_t ErrorCode`

Field Documentation

- **`USART_TypeDef* IRDA_HandleTypeDef::Instance`**
IRDA registers base address
- **`IRDA_InitTypeDef IRDA_HandleTypeDef::Init`**
IRDA communication parameters
- **`uint8_t* IRDA_HandleTypeDef::pTxBuffPtr`**
Pointer to IRDA Tx transfer Buffer
- **`uint16_t IRDA_HandleTypeDef::TxXferSize`**
IRDA Tx Transfer size
- **`__IO uint16_t IRDA_HandleTypeDef::TxXferCount`**
IRDA Tx Transfer Counter
- **`uint8_t* IRDA_HandleTypeDef::pRxBuffPtr`**
Pointer to IRDA Rx transfer Buffer
- **`uint16_t IRDA_HandleTypeDef::RxXferSize`**
IRDA Rx Transfer size
- **`__IO uint16_t IRDA_HandleTypeDef::RxXferCount`**
IRDA Rx Transfer Counter
- **`uint16_t IRDA_HandleTypeDef::Mask`**
IRDA RX RDR register mask
- **`DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx`**
IRDA Tx DMA Handle parameters
- **`DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx`**
IRDA Rx DMA Handle parameters
- **`HAL_LockTypeDef IRDA_HandleTypeDef::Lock`**
Locking object
- **`__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::gState`**
IRDA state information related to global Handle management and also related to Tx operations. This parameter can be a value of `HAL_IRDA_StateTypeDef`
- **`__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::RxState`**
IRDA state information related to Rx operations. This parameter can be a value of `HAL_IRDA_StateTypeDef`
- **`uint32_t IRDA_HandleTypeDef::ErrorCode`**
IRDA Error code

37.2 IRDA Firmware driver API description

37.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure (eg. IRDA_HandleTypeDef hirda).
2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API in setting the associated USART or UART in IRDA mode:
 - Enable the USARTx/UARTx interface clock.
 - USARTx/UARTx pins configuration:
 - Enable the clock for the USARTx/UARTx GPIOs.
 - Configure these USARTx/UARTx pins (TX as alternate function pull-up, RX as alternate function Input).
 - NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
 - Configure the USARTx/UARTx interrupt priority.
 - Enable the NVIC USARTx/UARTx IRQ handle.
 - The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
 - DMA Configuration if you need to use DMA process (HAL_IRDA_Transmit_DMA() and HAL_IRDA_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler in the hirda handle Init structure.
4. Initialize the IRDA registers by calling the HAL_IRDA_Init() API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_IRDA_MspInit() API.

Note:

The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.

5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission half of transfer HAL_IRDA_TxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception half of transfer HAL_IRDA_RxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

IRDA HAL driver macros list

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

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether or not the specified IRDA interrupt is enabled

Note:

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

37.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
 - Power mode
 - Prescaler setting
 - Receiver/transmitter modes

The HAL_IRDA_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [HAL_IRDA_Init](#)
- [HAL_IRDA_DelInit](#)
- [HAL_IRDA_MspInit](#)
- [HAL_IRDA_MspDelInit](#)

37.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA data transfers.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be

encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
 - Blocking mode: the communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - Non-Blocking mode: the communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
 - HAL_IRDA_Transmit()
 - HAL_IRDA_Receive()
3. Non Blocking mode APIs with Interrupt are :
 - HAL_IRDA_Transmit_IT()
 - HAL_IRDA_Receive_IT()
 - HAL_IRDA_IRQHandler()
4. Non Blocking mode functions with DMA are :
 - HAL_IRDA_Transmit_DMA()
 - HAL_IRDA_Receive_DMA()
 - HAL_IRDA_DMAPause()
 - HAL_IRDA_DMAResume()
 - HAL_IRDA_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:
 - HAL_IRDA_TxHalfCpltCallback()
 - HAL_IRDA_TxCpltCallback()
 - HAL_IRDA_RxHalfCpltCallback()
 - HAL_IRDA_RxCpltCallback()
 - HAL_IRDA_ErrorCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's : (+) HAL_IRDA_Abort() (+) HAL_IRDA_AbortTransmit() (+) HAL_IRDA_AbortReceive() (+) HAL_IRDA_Abort_IT() (+) HAL_IRDA_AbortTransmit_IT() (+) HAL_IRDA_AbortReceive_IT()
7. For Abort services based on interrupts (HAL_IRDA_Abortxxx_IT), a set of Abort Complete Callbacks are provided: (+) HAL_IRDA_AbortCpltCallback() (+) HAL_IRDA_AbortTransmitCpltCallback() (+) HAL_IRDA_AbortReceiveCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
 - (+) Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed. Transfer is kept ongoing on IRDA side. If user wants to abort it, Abort services should be called by user. (+) Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted. (#) There are two modes of transfer: (++) Blocking mode: the communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer. (++) Non-Blocking mode: the communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The

HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected (#) Blocking mode APIs are : (++) HAL_IRDA_Transmit() (++) HAL_IRDA_Receive() (#) Non Blocking mode APIs with Interrupt are : (++) HAL_IRDA_Transmit_IT() (++) HAL_IRDA_Receive_IT() (++) HAL_IRDA_IRQHandler() (#) Non Blocking mode functions with DMA are : (++) HAL_IRDA_Transmit_DMA() (++) HAL_IRDA_Receive_DMA() (++) HAL_IRDA_DMAPause() (++) HAL_IRDA_DMAResume() (++) HAL_IRDA_DMAStop() (#) A set of Transfer Complete Callbacks are provided in Non Blocking mode: (++) HAL_IRDA_TxHalfCpltCallback() (++) HAL_IRDA_TxCpltCallback() (++) HAL_IRDA_RxHalfCpltCallback() (++) HAL_IRDA_RxCpltCallback() (++) HAL_IRDA_ErrorCallback() (#) Non-Blocking mode transfers could be aborted using Abort API's :

- HAL_IRDA_Abort()
- HAL_IRDA_AbortTransmit()
- HAL_IRDA_AbortReceive()
- HAL_IRDA_Abort_IT()
- HAL_IRDA_AbortTransmit_IT()
- HAL_IRDA_AbortReceive_IT() (#) For Abort services based on interrupts (HAL_IRDA_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
- HAL_IRDA_AbortCpltCallback()
- HAL_IRDA_AbortTransmitCpltCallback()
- HAL_IRDA_AbortReceiveCpltCallback() (#) In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
- Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed. Transfer is kept ongoing on IRDA side. If user wants to abort it, Abort services should be called by user.
- Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed.

This section contains the following APIs:

- [HAL_IRDA_Transmit](#)
- [HAL_IRDA_Receive](#)
- [HAL_IRDA_Transmit_IT](#)
- [HAL_IRDA_Receive_IT](#)
- [HAL_IRDA_Transmit_DMA](#)
- [HAL_IRDA_Receive_DMA](#)
- [HAL_IRDA_DMAPause](#)
- [HAL_IRDA_DMAResume](#)
- [HAL_IRDA_DMAStop](#)
- [HAL_IRDA_Abort](#)
- [HAL_IRDA_AbortTransmit](#)
- [HAL_IRDA_AbortReceive](#)
- [HAL_IRDA_Abort_IT](#)
- [HAL_IRDA_AbortTransmit_IT](#)
- [HAL_IRDA_AbortReceive_IT](#)
- [HAL_IRDA_IRQHandler](#)
- [HAL_IRDA_TxHalfCpltCallback](#)
- [HAL_IRDA_TxCpltCallback](#)
- [HAL_IRDA_RxHalfCpltCallback](#)
- [HAL_IRDA_RxCpltCallback](#)
- [HAL_IRDA_ErrorCallback](#)
- [HAL_IRDA_AbortCpltCallback](#)
- [HAL_IRDA_AbortTransmitCpltCallback](#)

- [HAL_IRDA_AbortReceiveCpltCallback](#)

37.2.4 Peripheral State and Error functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IRDA peripheral handle.
- HAL_IRDA_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- [HAL_IRDA_GetState](#)
- [HAL_IRDA_GetError](#)

37.2.5 Detailed description of functions

HAL_IRDA_Init

Function name

`HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)`

Function description

Initialize the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and initialize the associated handle.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DeInit

Function name

`HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)`

Function description

DeInitialize the IRDA peripheral.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_MspInit

Function name

`void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)`

Function description

Initialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_MspDelInit

Function name

```
void HAL_IRDA_MspDelInit (IRDA_HandleTypeDef * hirda)
```

Function description

DeInitialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_Transmit

Function name

```
HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)
```

Function description

Send an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer.
- **Size:** Amount of data to be sent.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

HAL_IRDA_Receive

Function name

```
HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)
```

Function description

Receive an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer.
- **Size:** Amount of data to be received.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

HAL_IRDA_Transmit_IT

Function name

`HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)`

Function description

Send an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer.
- **Size:** Amount of data to be sent.

Return values

- **HAL:** status

HAL_IRDA_Receive_IT

Function name

`HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)`

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer.
- **Size:** Amount of data to be received.

Return values

- **HAL:** status

HAL_IRDA_Transmit_DMA

Function name

`HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)`

Function description

Send an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_IRDA_Receive_DMA

Function name

```
HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
```

Function description

Receive an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer.
- **Size:** Amount of data to be received.

Return values

- **HAL:** status

Notes

- When the IRDA parity is enabled (PCE = 1) the received data contains the parity bit (MSB position).

HAL_IRDA_DMAPause

Function name

```
HAL_StatusTypeDef HAL_IRDA_DM_PAUSE (IRDA_HandleTypeDef * hirda)
```

Function description

Pause the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DMAResume

Function name

```
HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)
```

Function description

Resume the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DMAStop

Function name

```
HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)
```

Function description

Stop the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_Abort

Function name

HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortTransmit

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortReceive

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_Abort_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name

void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)

Function description

Handle IRDA interrupt request.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxCpltCallback

Function name

void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_RxCpltCallback

Function name

void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxHalfCpltCallback

Function name

void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Half Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- **None:**

HAL_IRDA_RxHalfCpltCallback

Function name

void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Half Transfer complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_ErrorCallback

Function name

void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA error callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortCpltCallback

Function name

void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortTransmitCpltCallback

Function name

void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortReceiveCpltCallback

Function name

void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Receive Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_GetState

Function name

`HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)`

Function description

Return the IRDA handle state.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** state

HAL_IRDA_GetError

Function name

`uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)`

Function description

Return the IRDA handle error code.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **IRDA:** Error Code

37.3 IRDA Firmware driver defines

37.3.1 IRDA

IRDA DMA Rx

`IRDA_DMA_RX_DISABLE`

IRDA DMA RX disabled

`IRDA_DMA_RX_ENABLE`

IRDA DMA RX enabled

IRDA DMA Tx

`IRDA_DMA_TX_DISABLE`

IRDA DMA TX disabled

`IRDA_DMA_TX_ENABLE`

IRDA DMA TX enabled

IRDA Exported Macros

`_HAL_IRDA_RESET_HANDLE_STATE`

Description:

- Reset IRDA handle state.

Parameters:

- `__HANDLE__`: IRDA handle.

Return value:

- None

[__HAL_IRDA_FLUSH_DRREGISTER](#)**Description:**

- Flush the IRDA DR register.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

[__HAL_IRDA_CLEAR_FLAG](#)**Description:**

- Clear the specified IRDA pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be any combination of the following values:
 - `IRDA_CLEAR_PEF`
 - `IRDA_CLEAR_FEF`
 - `IRDA_CLEAR_NEF`
 - `IRDA_CLEAR_OREF`
 - `IRDA_CLEAR_TCF`
 - `IRDA_CLEAR_IDLEF`

Return value:

- None

[__HAL_IRDA_CLEAR_PEFLAG](#)**Description:**

- Clear the IRDA PE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

[__HAL_IRDA_CLEAR_FEFLAG](#)**Description:**

- Clear the IRDA FE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

[__HAL_IRDA_CLEAR_NEFLAG](#)**Description:**

- Clear the IRDA NE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

[`__HAL_IRDA_CLEAR_OREFLAG`](#)**Description:**

- Clear the IRDA ORE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

[`__HAL_IRDA_CLEAR_IDLEFLAG`](#)**Description:**

- Clear the IRDA IDLE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

[`__HAL_IRDA_GET_FLAG`](#)**Description:**

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

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `IRDA_FLAG_RXACK` Receive enable acknowledge flag
 - `IRDA_FLAG_TEACK` Transmit enable acknowledge flag
 - `IRDA_FLAG_BUSY` Busy flag
 - `IRDA_FLAG_ABRF` Auto Baud rate detection flag
 - `IRDA_FLAG_ABRE` Auto Baud rate detection error flag
 - `IRDA_FLAG_TXE` Transmit data register empty flag
 - `IRDA_FLAG_TC` Transmission Complete flag
 - `IRDA_FLAG_RXNE` Receive data register not empty flag
 - `IRDA_FLAG_ORE` OverRun Error flag
 - `IRDA_FLAG_NE` Noise Error flag
 - `IRDA_FLAG_FE` Framing Error flag
 - `IRDA_FLAG_PE` Parity Error flag

Return value:

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

[`__HAL_IRDA_ENABLE_IT`](#)**Description:**

- Enable the specified IRDA interrupt.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__INTERRUPT__`: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
 - `IRDA_IT_TXE` Transmit Data Register empty interrupt
 - `IRDA_IT_TC` Transmission complete interrupt
 - `IRDA_IT_RXNE` Receive Data register not empty interrupt
 - `IRDA_IT_IDLE` Idle line detection interrupt
 - `IRDA_IT_PE` Parity Error interrupt
 - `IRDA_IT_ERR` Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

[__HAL_IRDA_DISABLE_IT](#)**Description:**

- Disable the specified IRDA interrupt.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__INTERRUPT__`: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
 - `IRDA_IT_TXE` Transmit Data Register empty interrupt
 - `IRDA_IT_TC` Transmission complete interrupt
 - `IRDA_IT_RXNE` Receive Data register not empty interrupt
 - `IRDA_IT_IDLE` Idle line detection interrupt
 - `IRDA_IT_PE` Parity Error interrupt
 - `IRDA_IT_ERR` Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

[__HAL_IRDA_GET_IT](#)**Description:**

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__IT__`: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
 - `IRDA_IT_TXE` Transmit Data Register empty interrupt
 - `IRDA_IT_TC` Transmission complete interrupt
 - `IRDA_IT_RXNE` Receive Data register not empty interrupt
 - `IRDA_IT_IDLE` Idle line detection interrupt
 - `IRDA_IT_ORE` OverRun Error interrupt
 - `IRDA_IT_NE` Noise Error interrupt
 - `IRDA_IT_FE` Framing Error interrupt
 - `IRDA_IT_PE` Parity Error interrupt

Return value:

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

[__HAL_IRDA_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__IT__`: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
 - `IRDA_IT_TXE` Transmit Data Register empty interrupt
 - `IRDA_IT_TC` Transmission complete interrupt
 - `IRDA_IT_RXNE` Receive Data register not empty interrupt
 - `IRDA_IT_IDLE` Idle line detection interrupt
 - `IRDA_IT_ERR` Framing, overrun or noise error interrupt
 - `IRDA_IT_PE` Parity Error interrupt

Return value:

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

[__HAL_IRDA_CLEAR_IT](#)**Description:**

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__IT_CLEAR__`: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
 - `IRDA_CLEAR_PEF` Parity Error Clear Flag
 - `IRDA_CLEAR_FEF` Framing Error Clear Flag
 - `IRDA_CLEAR_NEF` Noise detected Clear Flag
 - `IRDA_CLEAR_OREF` OverRun Error Clear Flag
 - `IRDA_CLEAR_TCF` Transmission Complete Clear Flag

Return value:

- None

[__HAL_IRDA_SEND_REQ](#)**Description:**

- Set a specific IRDA request flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.
- `__REQ__`: specifies the request flag to set. This parameter can be one of the following values:
 - `IRDA_AUTOBAUD_REQUEST` Auto-Baud Rate Request
 - `IRDA_RXDATA_FLUSH_REQUEST` Receive Data flush Request
 - `IRDA_TXDATA_FLUSH_REQUEST` Transmit data flush Request

Return value:

- None

[__HAL_IRDA_ONE_BIT_SAMPLE_ENABLE](#)**Description:**

- Enable the IRDA one bit sample method.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

IRDA Flags

IRDA_FLAG_RXACK

IRDA Receive enable acknowledge flag

IRDA_FLAG_TEACK

IRDA Transmit enable acknowledge flag

IRDA_FLAG_BUSY

IRDA Busy flag

IRDA_FLAG_ABRF

IRDA Auto baud rate flag

IRDA_FLAG_ABRE

IRDA Auto baud rate error

IRDA_FLAG_TXE

IRDA Transmit data register empty

IRDA_FLAG_TC

IRDA Transmission complete

IRDA_FLAG_RXNE

IRDA Read data register not empty

IRDA_FLAG_ORE

IRDA Overrun error

IRDA_FLAG_NE

IRDA Noise error

IRDA_FLAG_FE

IRDA Framing error

IRDA_FLAG_PE

IRDA Parity error

IRDA interruptions flags mask**IRDA_IT_MASK**

IRDA Interruptions flags mask

IRDA Interrupts Definition**IRDA_IT_PE**

IRDA Parity error interruption

IRDA_IT_TXE

IRDA Transmit data register empty interruption

IRDA_IT_TC

IRDA Transmission complete interruption

IRDA_IT_RXNE

IRDA Read data register not empty interruption

IRDA_IT_IDLE

IRDA Idle interruption

IRDA_IT_ERR

IRDA Error interruption

IRDA_IT_ORE

IRDA Overrun error interruption

IRDA_IT_NE

IRDA Noise error interruption

IRDA_IT_FE

IRDA Frame error interruption

IRDA Interruption Clear Flags**IRDA_CLEAR_PEF**

Parity Error Clear Flag

IRDA_CLEAR_FEF

Framing Error Clear Flag

IRDA_CLEAR_NEF

Noise detected Clear Flag

IRDA_CLEAR_OREF

OverRun Error Clear Flag

IRDA_CLEAR_IDLEF

IDLE line detected Clear Flag

IRDA_CLEAR_TCF

Transmission Complete Clear Flag

*IRDA Low Power***IRDA_POWERMODE_NORMAL**

IRDA normal power mode

IRDA_POWERMODE_LOWPOWER

IRDA low power mode

*IRDA Mode***IRDA_MODE_DISABLE**

Associated UART disabled in IRDA mode

IRDA_MODE_ENABLE

Associated UART enabled in IRDA mode

*IRDA One Bit Sampling***IRDA_ONE_BIT_SAMPLE_DISABLE**

One-bit sampling disabled

IRDA_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enabled

*IRDA Parity***IRDA_PARITY_NONE**

No parity

IRDA_PARITY_EVEN

Even parity

IRDA_PARITY_ODD

Odd parity

*IRDA Request Parameters***IRDA_AUTOBAUD_REQUEST**

Auto-Baud Rate Request

IRDA_RXDATA_FLUSH_REQUEST

Receive Data flush Request

IRDA_TXDATA_FLUSH_REQUEST

Transmit data flush Request

IRDA State

IRDA_STATE_DISABLE

IRDA disabled

IRDA_STATE_ENABLE

IRDA enabled

IRDA Transfer Mode**IRDA_MODE_RX**

RX mode

IRDA_MODE_TX

TX mode

IRDA_MODE_TX_RX

RX and TX mode

IRDA Word Length**IRDA_WORDLENGTH_7B**

7-bit long frame

IRDA_WORDLENGTH_8B

8-bit long frame

IRDA_WORDLENGTH_9B

9-bit long frame

38 HAL IWDG Generic Driver

38.1 IWDG Firmware driver registers structures

38.1.1 IWDG_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t Reload*
- *uint32_t Window*

Field Documentation

- *uint32_t IWDG_InitTypeDef::Prescaler*

Select the prescaler of the IWDG. This parameter can be a value of **IWDG_Prescaler**

- *uint32_t IWDG_InitTypeDef::Reload*

Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFFFF

- *uint32_t IWDG_InitTypeDef::Window*

Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFFFF

38.1.2 IWDG_HandleTypeDef

Data Fields

- *IWDG_TypeDef * Instance*
- *IWDG_InitTypeDef Init*

Field Documentation

- *IWDG_TypeDef* IWDG_HandleTypeDef::Instance*

Register base address

- *IWDG_InitTypeDef IWDG_HandleTypeDef::Init*

IWDG required parameters

38.2 IWDG Firmware driver API description

38.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by Low-Speed clock (LSI) and thus stays active even if the main clock fails.
- Once the IWDG is started, the LSI is forced ON and both can not be disabled. The counter starts counting down from the reset value (0xFFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.

- Debug mode : When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG() and __HAL_DBGMCU_UNFREEZE_IWDG() macros

Min-max timeout value @32KHz (LSI): ~125us / ~32.7s The IWDG timeout may vary due to LSI frequency dispersion. STM32H7xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

38.2.2 How to use this driver

1. Use IWDG using HAL_IWDG_Init() function to :
 - Enable instance by writing Start keyword in IWDG_KEY register. LSI clock is forced ON and IWDG counter starts downcounting.
 - Enable write access to configuration register: IWDG_PR, IWDG_RLR & IWDG_WINR.
 - Configure the IWDG prescaler and counter reload value. This reload value will be loaded in the IWDG counter each time the watchdog is reloaded, then the IWDG will start counting down from this value.
 - Wait for status flags to be reset
 - Depending on window parameter:
 - If Window Init parameter is same as Window register value, nothing more is done but reload counter value in order to exit function with exact time base.
 - Else modify Window register. This will automatically reload watchdog counter.
2. Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

Below the list of most used macros in IWDG HAL driver:

- __HAL_IWDG_START: Enable the IWDG peripheral
- __HAL_IWDG_RELOAD_COUNTER: Reloads IWDG counter with value defined in the reload register

38.2.3 Initialization and Start functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef of associated handle.
- Manage Window option.
- Once initialization is performed in HAL_IWDG_Init function, Watchdog is reloaded in order to exit function with correct time base.

This section contains the following APIs:

- [HAL_IWDG_Init](#)

38.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- [HAL_IWDG_Refresh](#)

38.2.5 Detailed description of functions

HAL_IWDG_Init

Function name

HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)

Function description

Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and start watchdog.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

HAL_IWDG_Refresh

Function name

HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)

Function description

Refresh the IWDG.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

38.3 IWDG Firmware driver defines

38.3.1 IWDG

IWDG Exported Macros

[__HAL_IWDG_START](#)

Description:

- Enable the IWDG peripheral.

Parameters:

- [__HANDLE__](#): IWDG handle

Return value:

- None

[__HAL_IWDG_RELOAD_COUNTER](#)

Description:

- Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR & IWDG_WINR registers disabled).

Parameters:

- [__HANDLE__](#): IWDG handle

Return value:

- None

*IWDG Prescaler***IWDG_PRESCALER_4**

IWDG prescaler set to 4

IWDG_PRESCALER_8

IWDG prescaler set to 8

IWDG_PRESCALER_16

IWDG prescaler set to 16

IWDG_PRESCALER_32

IWDG prescaler set to 32

IWDG_PRESCALER_64

IWDG prescaler set to 64

IWDG_PRESCALER_128

IWDG prescaler set to 128

IWDG_PRESCALER_256

IWDG prescaler set to 256

*IWDG Window option***IWDG_WINDOW_DISABLE**

39 HAL JPEG Generic Driver

39.1 JPEG Firmware driver registers structures

39.1.1 JPEG_ConfTypeDef

Data Fields

- `uint8_t ColorSpace`
- `uint8_t ChromaSubsampling`
- `uint32_t ImageHeight`
- `uint32_t ImageWidth`
- `uint8_t ImageQuality`

Field Documentation

- `uint8_t JPEG_ConfTypeDef::ColorSpace`

Image Color space : gray-scale, YCBCR, RGB or CMYK This parameter can be a value of **JPEG ColorSpace**

- `uint8_t JPEG_ConfTypeDef::ChromaSubsampling`

Chroma Subsampling in case of YCBCR or CMYK color space, 0-> 4:4:4 , 1-> 4:2:2, 2 -> 4:1:1, 3 -> 4:2:0
This parameter can be a value of **JPEG Chrominance Sampling**

- `uint32_t JPEG_ConfTypeDef::ImageHeight`

Image height : number of lines

- `uint32_t JPEG_ConfTypeDef::ImageWidth`

Image width : number of pixels per line

- `uint8_t JPEG_ConfTypeDef::ImageQuality`

Quality of the JPEG encoding : from 1 to 100

39.1.2 JPEG_HandleTypeDef

Data Fields

- `JPEG_TypeDef * Instance`
- `JPEG_ConfTypeDef Conf`
- `uint8_t * pJpegInBuffPtr`
- `uint8_t * pJpegOutBuffPtr`
- `__IO uint32_t JpegInCount`
- `__IO uint32_t JpegOutCount`
- `uint32_t InDataLength`
- `uint32_t OutDataLength`
- `MDMA_HandleTypeDef * hdmain`
- `MDMA_HandleTypeDef * hdmaout`
- `uint8_t CustomQuanTable`
- `uint8_t * QuantTable0`
- `uint8_t * QuantTable1`
- `uint8_t * QuantTable2`
- `uint8_t * QuantTable3`
- `HAL_LockTypeDef Lock`

- `__IO HAL_JPEG_STATETypeDef State`
- `__IO uint32_t ErrorCode`
- `__IO uint32_t Context`

Field Documentation

- `JPEG_TypeDef* JPEG_HandleTypeDef::Instance`
JPEG peripheral register base address
- `JPEG_ConfTypeDef JPEG_HandleTypeDef::Conf`
Current JPEG encoding/decoding parameters
- `uint8_t* JPEG_HandleTypeDef::pJpegInBuffPtr`
Pointer to JPEG processing (encoding, decoding,...) input buffer
- `uint8_t* JPEG_HandleTypeDef::pJpegOutBuffPtr`
Pointer to JPEG processing (encoding, decoding,...) output buffer
- `__IO uint32_t JPEG_HandleTypeDef::JpegInCount`
Internal Counter of input data
- `__IO uint32_t JPEG_HandleTypeDef::JpegOutCount`
Internal Counter of output data
- `uint32_t JPEG_HandleTypeDef::InDataLength`
Input Buffer Length in Bytes
- `uint32_t JPEG_HandleTypeDef::OutDataLength`
Output Buffer Length in Bytes
- `MDMA_HandleTypeDef* JPEG_HandleTypeDef::hdmain`
JPEG In MDMA handle parameters
- `MDMA_HandleTypeDef* JPEG_HandleTypeDef::hdmaout`
JPEG Out MDMA handle parameters
- `uint8_t JPEG_HandleTypeDef::CustomQuanTable`
If set to 1 specify that user customized quantization tables are used
- `uint8_t* JPEG_HandleTypeDef::QuantTable0`
Basic Quantization Table for component 0
- `uint8_t* JPEG_HandleTypeDef::QuantTable1`
Basic Quantization Table for component 1
- `uint8_t* JPEG_HandleTypeDef::QuantTable2`
Basic Quantization Table for component 2
- `uint8_t* JPEG_HandleTypeDef::QuantTable3`
Basic Quantization Table for component 3
- `HAL_LockTypeDef JPEG_HandleTypeDef::Lock`
JPEG locking object
- `__IO HAL_JPEG_STATETypeDef JPEG_HandleTypeDef::State`
JPEG peripheral state
- `__IO uint32_t JPEG_HandleTypeDef::ErrorCode`
JPEG Error code
- `__IO uint32_t JPEG_HandleTypeDef::Context`

JPEG Internal context

39.2 JPEG Firmware driver API description

39.2.1 How to use this driver

1. Initialize the JPEG peripheral using HAL_JPEG_Init : No initialization parameters are required. Only the call to HAL_JPEG_Init is necessary to initialize the JPEG peripheral.
2. If operation is JPEG encoding use function HAL_JPEG_ConfigEncoding to set the encoding parameters (mandatory before calling the encoding function). the application can change the encoding parameter "ImageQuality" from 1 to 100 to obtain a more or less quality (visual quality vs the original row image), and inversely more or less jpg file size.
3. Note that for decoding operation the JPEG peripheral output data are organized in YCbCr blocks called MCU (Minimum Coded Unit) as defined in the JPEG specification ISO/IEC 10918-1 standard. It is up to the application to transform these YCbCr blocks to RGB data that can be displayed. Respectively, for Encoding operation the JPEG peripheral input should be organized in YCbCr MCU blocks. It is up to the application to perform the necessary RGB to YCbCr MCU blocks transformation before feeding the JPEG peripheral with data.
4. Use functions HAL_JPEG_Encode and HAL_JPEG_Decode to start respectively a JPEG encoding/decoding operation in polling method (blocking).
5. Use functions HAL_JPEG_Encode_IT and HAL_JPEG_Decode_IT to start respectively a JPEG encoding/decoding operation with Interrupt method (not blocking).
6. Use functions HAL_JPEG_Encode_DMA and HAL_JPEG_Decode_DMA to start respectively a JPEG encoding/decoding operation with DMA method (not blocking).
7. Callback HAL_JPEG_InfoReadyCallback is asserted if the current operation is a JPEG decoding to provide the application with JPEG image parameters. This callback is asserted when the JPEG peripheral successfully parses the JPEG header.
8. Callback HAL_JPEG_GetDataCallback is asserted for both encoding and decoding operations to inform the application that the input buffer has been consumed by the peripheral and to ask for a new data chunk if the operation (encoding/decoding) has not been complete yet.
 - This CallBack should be implemented in the application side. It should call the function HAL_JPEG_ConfigInputBuffer if new input data are available, or call HAL_JPEG_Pause with parameter XferSelection set to JPEG_PAUSE_RESUME_INPUT to inform the JPEG HAL driver that the ongoing operation shall pause waiting for the application to provide a new input data chunk. Once the application succeeds in getting new data and if the input has been paused, the application can call the function HAL_JPEG_ConfigInputBuffer to set the new input buffer and size, then resume the JPEG HAL input by calling new function HAL_JPEG_Resume. If the application has ended feeding the HAL JPEG with input data (no more input data), the application should call the function HAL_JPEG_ConfigInputBuffer (within the callback HAL_JPEG_GetDataCallback) with the parameter InDataLength set to zero.
 - The mechanism of HAL_JPEG_ConfigInputBuffer/HAL_JPEG_Pause/HAL_JPEG_Resume allows to the application to provide the input data (for encoding or decoding) by chunks. If the new input data chunk is not available (because data should be read from an input file for example) the application can pause the JPEG input (using function HAL_JPEG_Pause) Once the new input data chunk is available (read from a file for example), the application can call the function HAL_JPEG_ConfigInputBuffer to provide the HAL with the new chunk then resume the JPEG HAL input by calling function HAL_JPEG_Resume.
 - The application can call functions HAL_JPEG_ConfigInputBuffer then HAL_JPEG_Resume at any time (outside the HAL_JPEG_GetDataCallback) Once the new input chunk data is available. However, to keep data coherency, the function HAL_JPEG_Pause must be imperatively called (if necessary) within the callback HAL_JPEG_GetDataCallback, i.e. when the HAL JPEG has ended transferring the previous chunk buffer to the JPEG peripheral.
9. Callback HAL_JPEG_DataReadyCallback is asserted when the HAL JPEG driver has filled the given output buffer with the given size.
 - This CallBack should be implemented in the application side. It should call the function HAL_JPEG_ConfigOutputBuffer to provide the HAL JPEG driver with the new output buffer location and size to be used to store next data chunk. If the application is not ready to provide the output chunk

location then it can call the function HAL_JPEG_Pause with parameter XferSelection set to "JPEG_PAUSE_RESUME_OUTPUT" to inform the JPEG HAL driver that it shall pause output data. Once the application is ready to receive the new data chunk (output buffer location free or available) it should call the function HAL_JPEG_ConfigOutputBuffer to provide the HAL JPEG driver with the new output chunk buffer location and size, then call "HAL_JPEG_Resume" to inform the HAL that it shall resume outputting data in the given output buffer.

- The mechanism of HAL_JPEG_ConfigOutputBuffer/HAL_JPEG_Pause/HAL_JPEG_Resume allows the application to receive data from the JPEG peripheral by chunks. when a chunk is received, the application can pause the HAL JPEG output data to be able to process these received data (YCbCr to RGB conversion in case of decoding or data storage in case of encoding).
 - The application can call functions HAL_JPEG_ConfigOutputBuffer then HAL_JPEG_Resume. any time (outside the HAL_JPEG_DataReadyCallback) Once the output data buffer is free to use. However, to keep data coherency, the function HAL_JPEG_Pause must be imperatively called (if necessary) within the callback HAL_JPEG_DataReadyCallback, i.e when the HAL JPEG has ended Transferring the previous chunk buffer from the JPEG peripheral to the application.
10. Callback HAL_JPEG_EncodeCpltCallback is asserted when the HAL JPEG driver has ended the current JPEG encoding operation, and all output data has been transmitted to the application.
 11. Callback HAL_JPEG_DecodeCpltCallback is asserted when the HAL JPEG driver has ended the current JPEG decoding operation. and all output data has been transmitted to the application.
 12. Callback HAL_JPEG_ErrorCallback is asserted when an error occurred during the current operation. the application can call the function "HAL_JPEG_GetError" to retrieve the error codes.
 13. By default the HAL JPEG driver uses the default quantization tables as provide in the JPEG specification (ISO/IEC 10918-1 standard) for encoding. User can change these default tables if necessary using the function HAL_JPEG_SetUserQuantTables Note that for decoding the quantization tables are automatically extracted from the JPEG header.
 14. To control JPEG state you can use the following function: HAL_JPEG_GetState()

JPEG HAL driver macros list

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

- `_HAL_JPEG_RESET_HANDLE_STATE` : Reset JPEG handle state.
- `_HAL_JPEG_ENABLE` : Enable the JPEG peripheral.
- `_HAL_JPEG_DISABLE` : Disable the JPEG peripheral.
- `_HAL_JPEG_GET_FLAG` : Check the specified JPEG status flag.
- `_HAL_JPEG_CLEAR_FLAG` : Clear the specified JPEG status flag.
- `_HAL_JPEG_ENABLE_IT` : Enable the specified JPEG Interrupt.
- `_HAL_JPEG_DISABLE_IT` : Disable the specified JPEG Interrupt.
- `_HAL_JPEG_GET_IT_SOURCE` : returns the state of the specified JPEG Interrupt (Enabled or disabled).

39.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the JPEG peripheral and creates the associated handle
- Deinitialize the JPEG peripheral

This section contains the following APIs:

- [**HAL_JPEG_Init**](#)
- [**HAL_JPEG_DelInit**](#)
- [**HAL_JPEG_MspInit**](#)
- [**HAL_JPEG_MspDelInit**](#)

39.2.3 Configuration functions

This section provides functions allowing to:

- `HAL_JPEG_ConfigEncoding()` : JPEG encoding configuration

- HAL_JPEG_GetInfo() : Extract the image configuration from the JPEG header during the decoding
- HAL_JPEG_EnableHeaderParsing() : Enable JPEG Header parsing for decoding
- HAL_JPEG_DisableHeaderParsing() : Disable JPEG Header parsing for decoding
- HAL_JPEG_SetUserQuantTables : Modify the default Quantization tables used for JPEG encoding.

This section contains the following APIs:

- [HAL_JPEG_ConfigEncoding](#)
- [HAL_JPEG_GetInfo](#)
- [HAL_JPEG_EnableHeaderParsing](#)
- [HAL_JPEG_DisableHeaderParsing](#)
- [HAL_JPEG_SetUserQuantTables](#)

39.2.4 JPEG processing functions

This section provides functions allowing to:

- HAL_JPEG_Encode() : JPEG encoding with polling process
- HAL_JPEG_Decode() : JPEG decoding with polling process
- HAL_JPEG_Encode_IT() : JPEG encoding with interrupt process
- HAL_JPEG_Decode_IT() : JPEG decoding with interrupt process
- HAL_JPEG_Encode_DMA() : JPEG encoding with DMA process
- HAL_JPEG_Decode_DMA() : JPEG decoding with DMA process
- HAL_JPEG_Pause() : Pause the Input/Output processing
- HAL_JPEG_Resume() : Resume the JPEG Input/Output processing
- HAL_JPEG_ConfigInputBuffer() : Config Encoding/Decoding Input Buffer
- HAL_JPEG_ConfigOutputBuffer() : Config Encoding/Decoding Output Buffer
- HAL_JPEG_Abort() : Aborts the JPEG Encoding/Decoding

This section contains the following APIs:

- [HAL_JPEG_Encode](#)
- [HAL_JPEG_Decode](#)
- [HAL_JPEG_Encode_IT](#)
- [HAL_JPEG_Decode_IT](#)
- [HAL_JPEG_Encode_DMA](#)
- [HAL_JPEG_Decode_DMA](#)
- [HAL_JPEG_Pause](#)
- [HAL_JPEG_Resume](#)
- [HAL_JPEG_ConfigInputBuffer](#)
- [HAL_JPEG_ConfigOutputBuffer](#)
- [HAL_JPEG_Abort](#)

39.2.5 JPEG Decode and Encode callback functions

This section provides callback functions:

- HAL_JPEG_InfoReadyCallback() : Decoding JPEG Info ready callback
- HAL_JPEG_EncodeCpltCallback() : Encoding complete callback.
- HAL_JPEG_DecodeCpltCallback() : Decoding complete callback.
- HAL_JPEG_ErrorCallback() : JPEG error callback.
- HAL_JPEG_GetDataCallback() : Get New Data chunk callback.
- HAL_JPEG_DataReadyCallback() : Decoded/Encoded Data ready callback.

This section contains the following APIs:

- [*HAL_JPEG_InfoReadyCallback*](#)
- [*HAL_JPEG_EncodeCpltCallback*](#)
- [*HAL_JPEG_DecodeCpltCallback*](#)
- [*HAL_JPEG_ErrorCallback*](#)
- [*HAL_JPEG_GetDataCallback*](#)
- [*HAL_JPEG_DataReadyCallback*](#)

39.2.6 JPEG IRQ handler management

This section provides JPEG IRQ handler function.

- `HAL_JPEG_IRQHandler()` : handles JPEG interrupt request

This section contains the following APIs:

- [*HAL_JPEG_IRQHandler*](#)

39.2.7 Peripheral State and Error functions

This section provides JPEG State and Errors function.

- `HAL_JPEG_GetState()` : permits to get in run-time the JPEG state.
- `HAL_JPEG_GetError()` : Returns the JPEG error code if any.

This section contains the following APIs:

- [*HAL_JPEG_GetState*](#)
- [*HAL_JPEG_GetError*](#)

39.2.8 Detailed description of functions

`HAL_JPEG_Init`

Function name

`HAL_StatusTypeDef HAL_JPEG_Init (JPEG_HandleTypeDef * hjpeg)`

Function description

Initializes the JPEG according to the specified parameters in the `JPEG_InitTypeDef` and creates the associated handle.

Parameters

- **hjpeg**: pointer to a `JPEG_HandleTypeDef` structure that contains the configuration information for JPEG module

Return values

- **HAL**: status

`HAL_JPEG_DeInit`

Function name

`HAL_StatusTypeDef HAL_JPEG_DeInit (JPEG_HandleTypeDef * hjpeg)`

Function description

DeInitializes the JPEG peripheral.

Parameters

- **hjpeg**: pointer to a `JPEG_HandleTypeDef` structure that contains the configuration information for JPEG module

Return values

- **HAL:** status

HAL_JPEG_MspInit**Function name**

```
void HAL_JPEG_MspInit (JPEG_HandleTypeDef * hjpeg)
```

Function description

Initializes the JPEG MSP.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_MspDeInit**Function name**

```
void HAL_JPEG_MspDeInit (JPEG_HandleTypeDef * hjpeg)
```

Function description

DeInitializes JPEG MSP.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_ConfigEncoding**Function name**

```
HAL_StatusTypeDef HAL_JPEG_ConfigEncoding (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pConf)
```

Function description

Set the JPEG encoding configuration.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pConf:** pointer to a JPEG_ConfTypeDef structure that contains the encoding configuration

Return values

- **HAL:** status

HAL_JPEG_GetInfo**Function name**

```
HAL_StatusTypeDef HAL_JPEG_GetInfo (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pInfo)
```

Function description

Extract the image configuration from the JPEG header during the decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pInfo:** pointer to a JPEG_ConfTypeDef structure that contains The JPEG decoded header informations

Return values

- **HAL:** status

HAL_JPEG_EnableHeaderParsing**Function name**

HAL_StatusTypeDef HAL_JPEG_EnableHeaderParsing (JPEG_HandleTypeDef * hjpeg)

Function description

Enable JPEG Header parsing for decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the JPEG.

Return values

- **HAL:** status

HAL_JPEG_DisableHeaderParsing**Function name**

HAL_StatusTypeDef HAL_JPEG_DisableHeaderParsing (JPEG_HandleTypeDef * hjpeg)

Function description

Disable JPEG Header parsing for decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the JPEG.

Return values

- **HAL:** status

HAL_JPEG_SetUserQuantTables**Function name**

HAL_StatusTypeDef HAL_JPEG_SetUserQuantTables (JPEG_HandleTypeDef * hjpeg, uint8_t * QTable0, uint8_t * QTable1, uint8_t * QTable2, uint8_t * QTable3)

Function description

Modify the default Quantization tables used for JPEG encoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **QTable0:** : pointer to uint8_t , define the user quantification table for color component 1. If NULL assume no need to update the table and no error return
- **QTable1:** : pointer to uint8_t , define the user quantification table for color component 2. If NULL assume no need to update the table and no error return.
- **QTable2:** : pointer to uint8_t , define the user quantification table for color component 3, If NULL assume no need to update the table and no error return.

- **QTable3:** : pointer to uint8_t , define the user quantification table for color component 4. If NULL assume no need to update the table and no error return.

Return values

- **HAL:** status

HAL_JPEG_Encode

Function name

```
HAL_StatusTypeDef HAL_JPEG_Encode (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength, uint32_t Timeout)
```

Function description

Starts JPEG encoding with polling processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_JPEG_Decode

Function name

```
HAL_StatusTypeDef HAL_JPEG_Decode (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength, uint32_t Timeout)
```

Function description

Starts JPEG decoding with polling processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn:** Pointer to the input data buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOutMCU:** Pointer to the Output data buffer
- **OutDataLength:** size in bytes of the Output buffer
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_JPEG_Encode_IT

Function name

```
HAL_StatusTypeDef HAL_JPEG_Encode_IT (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength)
```

Function description

Starts JPEG encoding with interrupt processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

`HAL_JPEG_Decode_IT`

Function name

```
HAL_StatusTypeDef HAL_JPEG_Decode_IT (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength)
```

Function description

Starts JPEG decoding with interrupt processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn:** Pointer to the input data buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOutMCU:** Pointer to the Output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

`HAL_JPEG_Encode_DMA`

Function name

```
HAL_StatusTypeDef HAL_JPEG_Encode_DMA (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength)
```

Function description

Starts JPEG encoding with DMA processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Decode_DMA

Function name

```
HAL_StatusTypeDef HAL_JPEG_Decode_DMA (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength)
```

Function description

Starts JPEG decoding with DMA processing.

Parameters

- **hjpeg**: pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn**: Pointer to the input data buffer
- **InDataLength**: size in bytes Input buffer
- **pDataOutMCU**: Pointer to the Output data buffer
- **OutDataLength**: size in bytes of the Output buffer

Return values

- **HAL**: status

HAL_JPEG_Pause

Function name

```
HAL_StatusTypeDef HAL_JPEG_Pause (JPEG_HandleTypeDef * hjpeg, uint32_t XferSelection)
```

Function description

Pause the JPEG Input/Output processing.

Parameters

- **hjpeg**: pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **XferSelection**: This parameter can be one of the following values : JPEG_PAUSE_RESUME_INPUT : Pause Input processing JPEG_PAUSE_RESUME_OUTPUT: Pause Output processing JPEG_PAUSE_RESUME_INPUT_OUTPUT: Pause Input and Output processing

Return values

- **HAL**: status

HAL_JPEG_Resume

Function name

```
HAL_StatusTypeDef HAL_JPEG_Resume (JPEG_HandleTypeDef * hjpeg, uint32_t XferSelection)
```

Function description

Resume the JPEG Input/Output processing.

Parameters

- **hjpeg**: pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **XferSelection**: This parameter can be one of the following values : JPEG_PAUSE_RESUME_INPUT : Resume Input processing JPEG_PAUSE_RESUME_OUTPUT: Resume Output processing JPEG_PAUSE_RESUME_INPUT_OUTPUT: Resume Input and Output processing

Return values

- **HAL**: status

HAL_JPEG_ConfigInputBuffer

Function name

```
void HAL_JPEG_ConfigInputBuffer (JPEG_HandleTypeDef * hjpeg, uint8_t * pNewInputBuffer, uint32_t InDataLength)
```

Function description

Config Encoding/Decoding Input Buffer.

Parameters

- **hjpeg**: pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module.
- **pNewInputBuffer**: Pointer to the new input data buffer
- **InDataLength**: Size in bytes of the new Input data buffer

Return values

- **HAL**: status

HAL_JPEG_ConfigOutputBuffer

Function name

```
void HAL_JPEG_ConfigOutputBuffer (JPEG_HandleTypeDef * hjpeg, uint8_t * pNewOutputBuffer, uint32_t OutDataLength)
```

Function description

Config Encoding/Decoding Output Buffer.

Parameters

- **hjpeg**: pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module.
- **pNewOutputBuffer**: Pointer to the new output data buffer
- **OutDataLength**: Size in bytes of the new Output data buffer

Return values

- **HAL**: status

HAL_JPEG_Abort

Function name

```
HAL_StatusTypeDef HAL_JPEG_Abort (JPEG_HandleTypeDef * hjpeg)
```

Function description

Aborts the JPEG Encoding/Decoding.

Parameters

- **hjpeg**: pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **HAL**: status

HAL_JPEG_InfoReadyCallback

Function name

```
void HAL_JPEG_InfoReadyCallback (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pInfo)
```

Function description

Decoding JPEG Info ready callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pInfo:** pointer to a JPEG_ConfTypeDef structure that contains The JPEG decoded header informations

Return values

- **None:**

HAL_JPEG_EncodeCpltCallback

Function name

void HAL_JPEG_EncodeCpltCallback (JPEG_HandleTypeDef * hjpeg)

Function description

Encoding complete callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_DecodeCpltCallback

Function name

void HAL_JPEG_DecodeCpltCallback (JPEG_HandleTypeDef * hjpeg)

Function description

Decoding complete callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_ErrorCallback

Function name

void HAL_JPEG_ErrorCallback (JPEG_HandleTypeDef * hjpeg)

Function description

JPEG error callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_GetDataCallback

Function name

```
void HAL_JPEG_GetDataCallback (JPEG_HandleTypeDef * hjpeg, uint32_t NbDecodedData)
```

Function description

Get New Data chunk callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **NbDecodedData:** Number of consummed data in the previous chunk in bytes

Return values

- **None:**

HAL_JPEG_DataReadyCallback

Function name

```
void HAL_JPEG_DataReadyCallback (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataOut, uint32_t OutDataLength)
```

Function description

Decoded/Encoded Data ready callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataOut:** pointer to the output data buffer
- **OutDataLength:** number in bytes of data available in the specified output buffer

Return values

- **None:**

HAL_JPEG_IRQHandler

Function name

```
void HAL_JPEG_IRQHandler (JPEG_HandleTypeDef * hjpeg)
```

Function description

This function handles JPEG interrupt request.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_GetState

Function name

```
HAL_JPEG_STATETypeDef HAL_JPEG_GetState (JPEG_HandleTypeDef * hjpeg)
```

Function description

Returns the JPEG state.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **JPEG:** state

HAL_JPEG_GetError

Function name

`uint32_t HAL_JPEG_GetError (JPEG_HandleTypeDef * hjpeg)`

Function description

Return the JPEG error code.

Parameters

- **hjpeg:** : pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the specified JPEG.

Return values

- **JPEG:** Error Code

39.3 JPEG Firmware driver defines

39.3.1 JPEG

JPEG Chrominance Sampling

JPEG_444_SUBSAMPLING

Chroma Subsampling 4:4:4

JPEG_420_SUBSAMPLING

Chroma Subsampling 4:2:0

JPEG_422_SUBSAMPLING

Chroma Subsampling 4:2:2

JPEG ColorSpace

JPEG_GRAYSCALE_COLORSPACE

JPEG_YCBCR_COLORSPACE

JPEG_CMYK_COLORSPACE

JPEG Error Code definition

HAL_JPEG_ERROR_NONE

No error

HAL_JPEG_ERROR_HUFF_TABLE

Huffman Table programming error

HAL_JPEG_ERROR_QUANT_TABLE

Quantization Table programming error

HAL_JPEG_ERROR_DMA

DMA transfer error

HAL_JPEG_ERROR_TIMEOUT

Timeout error

JPEG Exported Macros

_HAL_JPEG_RESET_HANDLE_STATE

Description:

- Reset JPEG handle state.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

_HAL_JPEG_ENABLE

Description:

- Enable the JPEG peripheral.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

_HAL_JPEG_DISABLE

Description:

- Disable the JPEG peripheral.

Parameters:

- __HANDLE__: specifies the JPEG handle.

Return value:

- None

_HAL_JPEG_GET_FLAG

Description:

- Check the specified JPEG status flag.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __FLAG__: specifies the flag to check This parameter can be one of the following values:
 - JPEG_FLAG_IFFT : The input FIFO is not full and is bellow its threshold flag
 - JPEG_FLAG_IFNFF : The input FIFO Not Full Flag, a data can be written
 - JPEG_FLAG_OFTF : The output FIFO is not empty and has reach its threshold
 - JPEG_FLAG_OFNEF : The output FIFO is not empty, a data is available
 - JPEG_FLAG_EOCF : JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO
 - JPEG_FLAG_HPDF : JPEG Codec has finished the parsing of the headers and the internal registers have been updated
 - JPEG_FLAG_COF : JPEG Codec operation on going flag

Return value:

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

[__HAL_JPEG_CLEAR_FLAG](#)**Description:**

- Clear the specified JPEG status flag.

Parameters:

- __HANDLE__ : specifies the JPEG handle.
- __FLAG__ : specifies the flag to clear This parameter can be one of the following values:
 - JPEG_FLAG_EOCF : JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO
 - JPEG_FLAG_HPDF : JPEG Codec has finished the parsing of the headers

Return value:

- :: None

[__HAL_JPEG_ENABLE_IT](#)**Description:**

- Enable Interrupt.

Parameters:

- __HANDLE__ : specifies the JPEG handle.
- __INTERRUPT__ : specifies the interrupt to enable This parameter can be one of the following values:
 - JPEG_IT_IIFT : Input FIFO Threshold Interrupt
 - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
 - JPEG_IT_OIFT : Output FIFO Threshold Interrupt
 - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
 - JPEG_IT_EOC : End of Conversion Interrupt
 - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- :: No retrun

[__HAL_JPEG_DISABLE_IT](#)**Description:**

- Disable Interrupt.

Parameters:

- __HANDLE__ : specifies the JPEG handle.
- __INTERRUPT__ : specifies the interrupt to disable This parameter can be one of the following values:
 - JPEG_IT_IIFT : Input FIFO Threshold Interrupt
 - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
 - JPEG_IT_OIFT : Output FIFO Threshold Interrupt
 - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
 - JPEG_IT_EOC : End of Conversion Interrupt
 - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- :: No retrun

Notes:

- : To disable an IT we must use MODIFY_REG macro to avoid writing "1" to the FIFO flush bits located in the same IT enable register (CR register).

__HAL_JPEG_GET_IT_SOURCE

Description:

- Get Interrupt state.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to check This parameter can be one of the following values:
 - JPEG_IT_IIFT : Input FIFO Threshold Interrupt
 - JPEG_IT_IFNF : Input FIFO Not Full Interrupt
 - JPEG_IT_OFT : Output FIFO Threshold Interrupt
 - JPEG_IT_OFNE : Output FIFO Not empty Interrupt
 - JPEG_IT_EOC : End of Conversion Interrupt
 - JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- :: returns The new state of __INTERRUPT__ (Enabled or disabled)

JPEG Flag definition

JPEG_FLAG_IIFT

Input FIFO is not full and is bellow its threshold flag

JPEG_FLAG_IFNFF

Input FIFO Not Full Flag, a data can be written

JPEG_FLAG_OFTF

Output FIFO is not empty and has reach its threshold

JPEG_FLAG_OFNEF

Output FIFO is not empty, a data is available

JPEG_FLAG_EOCF

JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO

JPEG_FLAG_HPDF

JPEG Codec has finished the parsing of the headers and the internal registers have been updated

JPEG_FLAG_COF

JPEG Codec operation on going flag

JPEG_FLAG_ALL

JPEG Codec All previous flag

JPEG Image Quality

JPEG_IMAGE_QUALITY_MIN

Minimum JPEG quality

JPEG_IMAGE_QUALITY_MAX

Maximum JPEG quality

JPEG Interrupt configuration definition

JPEG_IT_IIFT

Input FIFO Threshold Interrupt

JPEG_IT_IFNF

Input FIFO Not Full Interrupt

JPEG_IT_OFT

Output FIFO Threshold Interrupt

JPEG_IT_OFNE

Output FIFO Not Empty Interrupt

JPEG_IT_EOC

End of Conversion Interrupt

JPEG_IT_HPD

Header Parsing Done Interrupt

JPEG Process Pause Resume definition**JPEG_PAUSE_RESUME_INPUT**

Pause/Resume Input FIFO Xfer

JPEG_PAUSE_RESUME_OUTPUT

Pause/Resume Output FIFO Xfer

JPEG_PAUSE_RESUME_INPUT_OUTPUT

Pause/Resume Input and Output FIFO Xfer

JPEG Quantization Table Size**JPEG_QUANT_TABLE_SIZE**

JPEG Quantization Table Size in bytes

40 HAL LPTIM Generic Driver

40.1 LPTIM Firmware driver registers structures

40.1.1 LPTIM_ClockConfigTypeDef

Data Fields

- `uint32_t Source`
- `uint32_t Prescaler`

Field Documentation

- `uint32_t LPTIM_ClockConfigTypeDef::Source`

Selects the clock source. This parameter can be a value of **LPTIM Clock Source**

- `uint32_t LPTIM_ClockConfigTypeDef::Prescaler`

Specifies the counter clock Prescaler. This parameter can be a value of **LPTIM Clock Prescaler**

40.1.2 LPTIM_ULPClockConfigTypeDef

Data Fields

- `uint32_t Polarity`
- `uint32_t SampleTime`

Field Documentation

- `uint32_t LPTIM_ULPClockConfigTypeDef::Polarity`

Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of **LPTIM Clock Polarity**

- `uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime`

Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of **LPTIM Clock Sample Time**

40.1.3 LPTIM_TriggerConfigTypeDef

Data Fields

- `uint32_t Source`
- `uint32_t ActiveEdge`
- `uint32_t SampleTime`

Field Documentation

- `uint32_t LPTIM_TriggerConfigTypeDef::Source`

Selects the Trigger source. This parameter can be a value of **LPTIM Trigger Source**

- `uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge`

Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of **LPTIM External Trigger Polarity**

- `uint32_t LPTIM_TriggerConfigTypeDef::SampleTime`

Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [LPTIM Trigger Sample Time](#)

40.1.4 LPTIM_InitTypeDef

Data Fields

- *LPTIM_ClockConfigTypeDef Clock*
- *LPTIM_ULPClockConfigTypeDef UltraLowPowerClock*
- *LPTIM_TriggerConfigTypeDef Trigger*
- *uint32_t OutputPolarity*
- *uint32_t UpdateMode*
- *uint32_t CounterSource*
- *uint32_t Input1Source*
- *uint32_t Input2Source*

Field Documentation

- *LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock*
Specifies the clock parameters
- *LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock*
Specifies the Ultra Low Power clock parameters
- *LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger*
Specifies the Trigger parameters
- *uint32_t LPTIM_InitTypeDef::OutputPolarity*
Specifies the Output polarity. This parameter can be a value of [LPTIM Output Polarity](#)
- *uint32_t LPTIM_InitTypeDef::UpdateMode*
Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of [LPTIM Updating Mode](#)
- *uint32_t LPTIM_InitTypeDef::CounterSource*
Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of [LPTIM Counter Source](#)
- *uint32_t LPTIM_InitTypeDef::Input1Source*
Specifies source selected for input1 (GPIO or comparator output). This parameter can be a value of [LPTIM Input1 Source](#)
- *uint32_t LPTIM_InitTypeDef::Input2Source*
Specifies source selected for input2 (GPIO or comparator output). Note: This parameter is used only for encoder feature so is used only for LPTIM1 instance. This parameter can be a value of [LPTIM Input2 Source](#)

40.1.5 LPTIM_HandleTypeDef

Data Fields

- *LPTIM_TypeDef * Instance*
- *LPTIM_InitTypeDef Init*
- *HAL_StatusTypeDef Status*
- *HAL_LockTypeDef Lock*
- *_IO HAL_LPTIM_StateTypeDef State*

Field Documentation

- *LPTIM_TypeDef* LPTIM_HandleTypeDef::Instance*

- Register base address
- *LPTIM_InitTypeDef LPTIM_HandleTypeDef::Init*
 - LPTIM required parameters
- *HAL_StatusTypeDef LPTIM_HandleTypeDef::Status*
 - LPTIM peripheral status
- *HAL_LockTypeDef LPTIM_HandleTypeDef::Lock*
 - LPTIM locking object
- *__IO HAL_LPTIM_StateTypeDef LPTIM_HandleTypeDef::State*
 - LPTIM peripheral state

40.2 LPTIM Firmware driver API description

40.2.1 How to use this driver

The LPTIM HAL driver can be used as follows:

1. Initialize the LPTIM low level resources by implementing the `HAL_LPTIM_MspInit()`:
 - Enable the LPTIM interface clock using `__HAL_RCC_LPTIMx_CLK_ENABLE()`.
 - In case of using interrupts (e.g. `HAL_LPTIM_PWM_Start_IT()`):
 - Configure the LPTIM interrupt priority using `HAL_NVIC_SetPriority()`.
 - Enable the LPTIM IRQ handler using `HAL_NVIC_EnableIRQ()`.
 - In LPTIM IRQ handler, call `HAL_LPTIM_IRQHandler()`.
2. Initialize the LPTIM HAL using `HAL_LPTIM_Init()`. This function configures mainly:
 - The instance: LPTIM1 or LPTIM2.
 - Clock: the counter clock.
 - Source : it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE, LSI or MSI).
 - Prescaler: select the clock divider.
 - UltraLowPowerClock : To be used only if the ULPTIM is selected as counter clock source.
 - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected.
 - SampleTime: clock sampling time to configure the clock glitch filter.
 - Trigger: How the counter start.
 - Source: trigger can be software or one of the hardware triggers.
 - ActiveEdge : only for hardware trigger.
 - SampleTime : trigger sampling time to configure the trigger glitch filter.
 - OutputPolarity : 2 opposite polarities are possible.
 - UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period.
 - Input1Source: Source selected for input1 (GPIO or comparator output).
 - Input2Source: Source selected for input2 (GPIO or comparator output). Input2 is used only for encoder feature so is used only for LPTIM1 instance.
3. Six modes are available:
 - PWM Mode: To generate a PWM signal with specified period and pulse, call `HAL_LPTIM_PWM_Start()` or `HAL_LPTIM_PWM_Start_IT()` for interruption mode.
 - One Pulse Mode: To generate pulse with specified width in response to a stimulus, call `HAL_LPTIM_OnePulse_Start()` or `HAL_LPTIM_OnePulse_Start_IT()` for interruption mode.
 - Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare match occurs. To start this mode, call `HAL_LPTIM_SetOnce_Start()` or `HAL_LPTIM_SetOnce_Start_IT()` for interruption mode.

- Encoder Mode: To use the encoder interface call HAL_LPTIM_Encoder_Start() or HAL_LPTIM_Encoder_Start_IT() for interruption mode. Only available for LPTIM1 instance.
 - Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call HAL_LPTIM_TimeOut_Start_IT() or HAL_LPTIM_TimeOut_Start_IT() for interruption mode.
 - Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL_LPTIM_Counter_Start() or HAL_LPTIM_Counter_Start_IT() for interruption mode.
4. User can stop any process by calling the corresponding API: HAL_LPTIM_Xxx_Stop() or HAL_LPTIM_Xxx_Stop_IT() if the process is already started in interruption mode.
 5. Deinitialize the LPTIM peripheral using HAL_LPTIM_Delinit().

40.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.
- Deinitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- Deinitialize the LPTIM MSP.

This section contains the following APIs:

- [**HAL_LPTIM_Init**](#)
- [**HAL_LPTIM_Delinit**](#)
- [**HAL_LPTIM_MspInit**](#)
- [**HAL_LPTIM_MspDelinit**](#)

40.2.3 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- [**HAL_LPTIM_PWM_Start**](#)
- [**HAL_LPTIM_PWM_Stop**](#)
- [**HAL_LPTIM_PWM_Start_IT**](#)
- [**HAL_LPTIM_PWM_Stop_IT**](#)
- [**HAL_LPTIM_OnePulse_Start**](#)
- [**HAL_LPTIM_OnePulse_Stop**](#)
- [**HAL_LPTIM_OnePulse_Start_IT**](#)

- [`HAL_LPTIM_OnePulse_Stop_IT`](#)
- [`HAL_LPTIM_SetOnce_Start`](#)
- [`HAL_LPTIM_SetOnce_Stop`](#)
- [`HAL_LPTIM_SetOnce_Start_IT`](#)
- [`HAL_LPTIM_SetOnce_Stop_IT`](#)
- [`HAL_LPTIM_Encoder_Start`](#)
- [`HAL_LPTIM_Encoder_Stop`](#)
- [`HAL_LPTIM_Encoder_Start_IT`](#)
- [`HAL_LPTIM_Encoder_Stop_IT`](#)
- [`HAL_LPTIM_TimeOut_Start`](#)
- [`HAL_LPTIM_TimeOut_Stop`](#)
- [`HAL_LPTIM_TimeOut_Start_IT`](#)
- [`HAL_LPTIM_TimeOut_Stop_IT`](#)
- [`HAL_LPTIM_Counter_Start`](#)
- [`HAL_LPTIM_Counter_Stop`](#)
- [`HAL_LPTIM_Counter_Start_IT`](#)
- [`HAL_LPTIM_Counter_Stop_IT`](#)

40.2.4 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare) value.

This section contains the following APIs:

- [`HAL_LPTIM_ReadCounter`](#)
- [`HAL_LPTIM_ReadAutoReload`](#)
- [`HAL_LPTIM_ReadCompare`](#)

40.2.5 LPTIM IRQ handler and callbacks

This section provides LPTIM IRQ handler and callback functions called within the IRQ handler.

This section contains the following APIs:

- [`HAL_LPTIM_IRQHandler`](#)
- [`HAL_LPTIM_CompareMatchCallback`](#)
- [`HAL_LPTIM_AutoReloadMatchCallback`](#)
- [`HAL_LPTIM_TriggerCallback`](#)
- [`HAL_LPTIM_CompareWriteCallback`](#)
- [`HAL_LPTIM_AutoReloadWriteCallback`](#)
- [`HAL_LPTIM_DirectionUpCallback`](#)
- [`HAL_LPTIM_DirectionDownCallback`](#)

40.2.6 Peripheral State functions

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

This section contains the following APIs:

- [`HAL_LPTIM_GetState`](#)

40.2.7 Detailed description of functions

HAL_LPTIM_Init

Function name

`HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hltim)`

Function description

Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.

Parameters

- **hltim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_DeInit

Function name

`HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hltim)`

Function description

DeInitialize the LPTIM peripheral.

Parameters

- **hltim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_MspInit

Function name

`void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hltim)`

Function description

Initialize the LPTIM MSP.

Parameters

- **hltim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_MspDeInit

Function name

`void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hltim)`

Function description

DeInitialize LPTIM MSP.

Parameters

- **hltim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM PWM generation.

Parameters

- **hltim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hltim)

Function description

Stop the LPTIM PWM generation.

Parameters

- **hltim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM PWM generation in interrupt mode.

Parameters

- **hltim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hltim)

Function description

Stop the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM One pulse generation.

Parameters

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the LPTIM One pulse generation.

Parameters

- **hlptim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)

Function description

Start the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop_IT

Function name

`HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hltim)`

Function description

Stop the LPTIM One pulse generation in interrupt mode.

Parameters

- **hltim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start

Function name

`HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)`

Function description

Start the LPTIM in Set once mode.

Parameters

- **hltim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop

Function name

`HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hltim)`

Function description

Stop the LPTIM Set once mode.

Parameters

- **hltim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start_IT

Function name

`HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)`

Function description

Start the LPTIM Set once mode in interrupt mode.

Parameters

- **hltim:** : LPTIM handle

- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Pulse:** : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * htim)

Function description

Stop the LPTIM Set once mode in interrupt mode.

Parameters

- **htim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * htim, uint32_t Period)

Function description

Starts the Encoder interface.

Parameters

- **htim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * htim)

Function description

Stop the Encoder interface.

Parameters

- **htim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * htim, uint32_t Period)

Function description

Start the Encoder interface in interrupt mode.

Parameters

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface in nterrupt mode.

Parameters

- **hlptim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Timeout)

Function description

Start the Timeout function.

Parameters

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Timeout:** : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function.

Parameters

- **hlptim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start_IT

Function name

```
HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period,  
uint32_t Timeout)
```

Function description

Start the Timeout function in interrupt mode.

Parameters

- **hltim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Timeout:** : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop_IT

Function name

```
HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hltim)
```

Function description

Stop the Timeout function in interrupt mode.

Parameters

- **hltim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start

Function name

```
HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period)
```

Function description

Start the Counter mode.

Parameters

- **hltim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop

Function name

```
HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hltim)
```

Function description

Stop the Counter mode.

Parameters

- **hiptim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hiptim, uint32_t Period)

Function description

Start the Counter mode in interrupt mode.

Parameters

- **hiptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hiptim)

Function description

Stop the Counter mode in interrupt mode.

Parameters

- **hiptim:** : LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_ReadCounter

Function name

uint32_t HAL_LPTIM_ReadCounter (LPTIM_HandleTypeDef * hiptim)

Function description

Return the current counter value.

Parameters

- **hiptim:** LPTIM handle

Return values

- **Counter:** value.

HAL_LPTIM_ReadAutoReload

Function name

uint32_t HAL_LPTIM_ReadAutoReload (LPTIM_HandleTypeDef * hiptim)

Function description

Return the current Autoreload (Period) value.

Parameters

- **hiptim:** LPTIM handle

Return values

- **Autoreload:** value.

HAL_LPTIM_ReadCompare

Function name

```
uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hiptim)
```

Function description

Return the current Compare (Pulse) value.

Parameters

- **hiptim:** LPTIM handle

Return values

- **Compare:** value.

HAL_LPTIM_IRQHandler

Function name

```
void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hiptim)
```

Function description

Handle LPTIM interrupt request.

Parameters

- **hiptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareMatchCallback

Function name

```
void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hiptim)
```

Function description

Compare match callback in non-blocking mode.

Parameters

- **hiptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadMatchCallback

Function name

```
void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hiptim)
```

Function description

Autoreload match callback in non-blocking mode.

Parameters

- **hiptim:** : LPTIM handle

Return values

- **None:**

HAL_LPTIM_TriggerCallback

Function name

void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hiptim)

Function description

Trigger detected callback in non-blocking mode.

Parameters

- **hiptim:** : LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareWriteCallback

Function name

void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hiptim)

Function description

Compare write callback in non-blocking mode.

Parameters

- **hiptim:** : LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadWriteCallback

Function name

void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hiptim)

Function description

Autoreload write callback in non-blocking mode.

Parameters

- **hiptim:** : LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionUpCallback

Function name

void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hiptim)

Function description

Direction counter changed from Down to Up callback in non-blocking mode.

Parameters

- **hiptim:** : LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionDownCallback

Function name

void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hiptim)

Function description

Direction counter changed from Up to Down callback in non-blocking mode.

Parameters

- **hiptim:** : LPTIM handle

Return values

- **None:**

HAL_LPTIM_GetState

Function name

HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hiptim)

Function description

Returns the LPTIM state.

Parameters

- **hiptim:** LPTIM handle

Return values

- **HAL:** state

40.3 LPTIM Firmware driver defines

40.3.1 LPTIM

LPTIM Clock Polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

LPTIM Clock Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

[LPTIM_PRESCALER_DIV8](#)

[LPTIM_PRESCALER_DIV16](#)

[LPTIM_PRESCALER_DIV32](#)

[LPTIM_PRESCALER_DIV64](#)

[LPTIM_PRESCALER_DIV128](#)

LPTIM Clock Sample Time

[LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION](#)

[LPTIM_CLOCKSAMPLETIME_2TRANSITIONS](#)

[LPTIM_CLOCKSAMPLETIME_4TRANSITIONS](#)

[LPTIM_CLOCKSAMPLETIME_8TRANSITIONS](#)

LPTIM Clock Source

[LPTIM_CLOCKSOURCE_APBCLOCK_LPOS](#)

[LPTIM_CLOCKSOURCE_ULPTIM](#)

LPTIM Counter Source

[LPTIM_COUNTERSOURCE_INTERNAL](#)

[LPTIM_COUNTERSOURCE_EXTERNAL](#)

LPTIM Exported Macros

[__HAL_LPTIM_RESET_HANDLE_STATE](#)

Description:

- Reset LPTIM handle state.

Parameters:

- `__HANDLE__`: LPTIM handle

Return value:

- None

[__HAL_LPTIM_ENABLE](#)

Description:

- Enable the LPTIM peripheral.

Parameters:

- `__HANDLE__`: LPTIM handle

Return value:

- None

[__HAL_LPTIM_DISABLE](#)

Description:

- Disable the LPTIM peripheral.

Parameters:

- `__HANDLE__`: LPTIM handle

Return value:

- None

[__HAL_LPTIM_START_CONTINUOUS](#)**Description:**

- Start the LPTIM peripheral in Continuous mode.

Parameters:

- `__HANDLE__`: LPTIM handle

Return value:

- None

[__HAL_LPTIM_START_SINGLE](#)**Description:**

- Start the LPTIM peripheral in single mode.

Parameters:

- `__HANDLE__`: LPTIM handle

Return value:

- None

[__HAL_LPTIM_RESET_COUNTER](#)**Description:**

- Reset the LPTIM Counter register in synchronous mode.

Parameters:

- `__HANDLE__`: LPTIM handle

Return value:

- None

[__HAL_LPTIM_RESET_COUNTER_AFTERREAD](#)**Description:**

- Reset after read of the LPTIM Counter register in asynchronous mode.

Parameters:

- `__HANDLE__`: LPTIM handle

Return value:

- None

[__HAL_LPTIM_AUTORELOAD_SET](#)**Description:**

- Write the passed parameter in the Autoreload register.

Parameters:

- `__HANDLE__`: LPTIM handle
- `__VALUE__`: Autoreload value

Return value:

- None

__HAL_LPTIM_COMPARE_SET

Description:

- Write the passed parameter in the Compare register.

Parameters:

- __HANDLE__: LPTIM handle
- __VALUE__: Compare value

Return value:

- None

__HAL_LPTIM_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: LPTIM handle
- __FLAG__: LPTIM flag to check This parameter can be a value of:
 - LPTIM_FLAG_DOWN : Counter direction change up Flag.
 - LPTIM_FLAG_UP : Counter direction change down to up Flag.
 - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
 - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
 - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
 - LPTIM_FLAG_ARRM : Autoreload match Flag.
 - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG

Description:

- Clear the specified LPTIM flag.

Parameters:

- __HANDLE__: LPTIM handle.
- __FLAG__: LPTIM flag to clear. This parameter can be a value of:
 - LPTIM_FLAG_DOWN : Counter direction change up Flag.
 - LPTIM_FLAG_UP : Counter direction change down to up Flag.
 - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
 - LPTIM_FLAG_CMPOK : Compare register update OK Flag.
 - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
 - LPTIM_FLAG_ARRM : Autoreload match Flag.
 - LPTIM_FLAG_CMPM : Compare match Flag.

Return value:

- None.

__HAL_LPTIM_ENABLE_IT

Description:

- Enable the specified LPTIM interrupt.

Parameters:

- __HANDLE__: LPTIM handle.

- `_INTERRUPT_`: LPTIM interrupt to set. This parameter can be a value of:
 - `LPTIM_IT_DOWN` : Counter direction change up Interrupt.
 - `LPTIM_IT_UP` : Counter direction change down to up Interrupt.
 - `LPTIM_IT_ARROK` : Autoreload register update OK Interrupt.
 - `LPTIM_IT_CMPOK` : Compare register update OK Interrupt.
 - `LPTIM_IT_EXTRIG` : External trigger edge event Interrupt.
 - `LPTIM_IT_ARRM` : Autoreload match Interrupt.
 - `LPTIM_IT_CMPM` : Compare match Interrupt.

Return value:

- None.

[`_HAL_LPTIM_DISABLE_IT`](#)**Description:**

- Disable the specified LPTIM interrupt.

Parameters:

- `_HANDLE_`: LPTIM handle.
- `_INTERRUPT_`: LPTIM interrupt to set. This parameter can be a value of:
 - `LPTIM_IT_DOWN` : Counter direction change up Interrupt.
 - `LPTIM_IT_UP` : Counter direction change down to up Interrupt.
 - `LPTIM_IT_ARROK` : Autoreload register update OK Interrupt.
 - `LPTIM_IT_CMPOK` : Compare register update OK Interrupt.
 - `LPTIM_IT_EXTRIG` : External trigger edge event Interrupt.
 - `LPTIM_IT_ARRM` : Autoreload match Interrupt.
 - `LPTIM_IT_CMPM` : Compare match Interrupt.

Return value:

- None.

[`_HAL_LPTIM_GET_IT_SOURCE`](#)**Description:**

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

Parameters:

- `_HANDLE_`: LPTIM handle.
- `_INTERRUPT_`: LPTIM interrupt to check. This parameter can be a value of:
 - `LPTIM_IT_DOWN` : Counter direction change up Interrupt.
 - `LPTIM_IT_UP` : Counter direction change down to up Interrupt.
 - `LPTIM_IT_ARROK` : Autoreload register update OK Interrupt.
 - `LPTIM_IT_CMPOK` : Compare register update OK Interrupt.
 - `LPTIM_IT_EXTRIG` : External trigger edge event Interrupt.
 - `LPTIM_IT_ARRM` : Autoreload match Interrupt.
 - `LPTIM_IT_CMPM` : Compare match Interrupt.

Return value:

- Interrupt: status.

LPTIM External Trigger Polarity[`LPTIM_ACTIVEEDGE_RISING`](#)[`LPTIM_ACTIVEEDGE_FALLING`](#)

`LPTIM_ACTIVEEDGE_RISING_FALLING`

LPTIM Flags Definition

`LPTIM_FLAG_DOWN`

`LPTIM_FLAG_UP`

`LPTIM_FLAG_ARROK`

`LPTIM_FLAG_CMPOK`

`LPTIM_FLAG_EXTTRIG`

`LPTIM_FLAG_ARRM`

`LPTIM_FLAG_CMPM`

LPTIM Input1 Source

`LPTIM_INPUT1SOURCE_GPIO`

For LPTIM1, LPTIM2 and LPTIM3

`LPTIM_INPUT1SOURCE_COMP1`

For LPTIM1 and LPTIM2

`LPTIM_INPUT1SOURCE_COMP2`

For LPTIM2 and LPTIM2

`LPTIM_INPUT1SOURCE_COMP1_COMP2`

For LPTIM2

`LPTIM_INPUT1SOURCE_SAI1_FSA`

For LPTIM3

`LPTIM_INPUT1SOURCE_SAI1_FSB`

For LPTIM3

LPTIM Input2 Source

`LPTIM_INPUT2SOURCE_GPIO`

For LPTIM1 and LPTIM2

`LPTIM_INPUT2SOURCE_COMP2`

For LPTIM1 and LPTIM2

LPTIM Interrupts Definition

`LPTIM_IT_DOWN`

`LPTIM_IT_UP`

`LPTIM_IT_ARROK`

`LPTIM_IT_CMPOK`

`LPTIM_IT_EXTTRIG`

LPTIM_IT_ARRM

LPTIM_IT_CMPM

LPTIM Output Polarity

LPTIM_OUTPUTPOLARITY_HIGH

LPTIM_OUTPUTPOLARITY_LOW

LPTIM Trigger Sample Time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION

LPTIM_TRIGSAMPLETIME_2TRANSITIONS

LPTIM_TRIGSAMPLETIME_4TRANSITIONS

LPTIM_TRIGSAMPLETIME_8TRANSITIONS

LPTIM Trigger Source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

LPTIM Updating Mode

LPTIM_UPDATE_IMMEDIATE

LPTIM_UPDATE_ENDOFPERIOD

41 HAL LTDC Generic Driver

41.1 LTDC Firmware driver registers structures

41.1.1 LTDC_ColorTypeDef

Data Fields

- *uint8_t Blue*
- *uint8_t Green*
- *uint8_t Red*
- *uint8_t Reserved*

Field Documentation

- *uint8_t LTDC_ColorTypeDef::Blue*

Configures the blue value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

- *uint8_t LTDC_ColorTypeDef::Green*

Configures the green value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

- *uint8_t LTDC_ColorTypeDef::Red*

Configures the red value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

- *uint8_t LTDC_ColorTypeDef::Reserved*

Reserved 0xFF

41.1.2 LTDC_InitTypeDef

Data Fields

- *uint32_t HSPolarity*
- *uint32_t VSPolarity*
- *uint32_t DEPolarity*
- *uint32_t PCPolarity*
- *uint32_t HorizontalSync*
- *uint32_t VerticalSync*
- *uint32_t AccumulatedHBP*
- *uint32_t AccumulatedVBP*
- *uint32_t AccumulatedActiveW*
- *uint32_t AccumulatedActiveH*
- *uint32_t TotalWidth*
- *uint32_t TotalHeight*
- *LTDC_ColorTypeDef Backcolor*

Field Documentation

- *uint32_t LTDC_InitTypeDef::HSPolarity*

configures the horizontal synchronization polarity. This parameter can be one value of **LTDC HS POLARITY**

- *uint32_t LTDC_InitTypeDef::VSPolarity*

configures the vertical synchronization polarity. This parameter can be one value of **LTDC VS POLARITY**

- ***uint32_t LTDC_InitTypeDef::DEPolarity***

configures the data enable polarity. This parameter can be one of value of **LTDC DE POLARITY**

- ***uint32_t LTDC_InitTypeDef::PCPolarity***

configures the pixel clock polarity. This parameter can be one of value of **LTDC PC POLARITY**

- ***uint32_t LTDC_InitTypeDef::HorizontalSync***

configures the number of Horizontal synchronization width. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.

- ***uint32_t LTDC_InitTypeDef::VerticalSync***

configures the number of Vertical synchronization height. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.

- ***uint32_t LTDC_InitTypeDef::AccumulatedHBP***

configures the accumulated horizontal back porch width. This parameter must be a number between Min_Data = LTDC_HorizontalSync and Max_Data = 0xFFFF.

- ***uint32_t LTDC_InitTypeDef::AccumulatedVBP***

configures the accumulated vertical back porch height. This parameter must be a number between Min_Data = LTDC_VerticalSync and Max_Data = 0x7FF.

- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveW***

configures the accumulated active width. This parameter must be a number between Min_Data = LTDC_AccumulatedHBP and Max_Data = 0xFFFF.

- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveH***

configures the accumulated active height. This parameter must be a number between Min_Data = LTDC_AccumulatedVBP and Max_Data = 0x7FF.

- ***uint32_t LTDC_InitTypeDef::TotalWidth***

configures the total width. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveW and Max_Data = 0xFFFF.

- ***uint32_t LTDC_InitTypeDef::TotalHeigh***

configures the total height. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveH and Max_Data = 0x7FF.

- ***LTDC_ColorTypeDef LTDC_InitTypeDef::Backcolor***

Configures the background color.

41.1.3 LTDC_LayerCfgTypeDef

Data Fields

- ***uint32_t WindowX0***
- ***uint32_t WindowX1***
- ***uint32_t WindowY0***
- ***uint32_t WindowY1***
- ***uint32_t PixelFormat***
- ***uint32_t Alpha***
- ***uint32_t Alpha0***
- ***uint32_t BlendingFactor1***
- ***uint32_t BlendingFactor2***
- ***uint32_t FBStartAdress***
- ***uint32_t ImageWidth***
- ***uint32_t ImageHeight***

- *LTDC_ColorTypeDef Backcolor*

Field Documentation

- *uint32_t LTDC_LayerCfgTypeDef::WindowX0*

Configures the Window Horizontal Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.

- *uint32_t LTDC_LayerCfgTypeDef::WindowX1*

Configures the Window Horizontal Stop Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.

- *uint32_t LTDC_LayerCfgTypeDef::WindowY0*

Configures the Window vertical Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.

- *uint32_t LTDC_LayerCfgTypeDef::WindowY1*

Configures the Window vertical Stop Position. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x7FF.

- *uint32_t LTDC_LayerCfgTypeDef::PixelFormat*

Specifies the pixel format. This parameter can be one of value of **LTDC Pixel format**

- *uint32_t LTDC_LayerCfgTypeDef::Alpha*

Specifies the constant alpha used for blending. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

- *uint32_t LTDC_LayerCfgTypeDef::Alpha0*

Configures the default alpha value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

- *uint32_t LTDC_LayerCfgTypeDef::BlendingFactor1*

Select the blending factor 1. This parameter can be one of value of **LTDC Blending Factor1**

- *uint32_t LTDC_LayerCfgTypeDef::BlendingFactor2*

Select the blending factor 2. This parameter can be one of value of **LTDC Blending Factor2**

- *uint32_t LTDC_LayerCfgTypeDef::FBStartAdress*

Configures the color frame buffer address

- *uint32_t LTDC_LayerCfgTypeDef::ImageWidth*

Configures the color frame buffer line length. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x1FFF.

- *uint32_t LTDC_LayerCfgTypeDef::ImageHeight*

Specifies the number of line in frame buffer. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.

- *LTDC_ColorTypeDef LTDC_LayerCfgTypeDef::Backcolor*

Configures the layer background color.

41.1.4 LTDC_HandleTypeDef

Data Fields

- *LTDC_TypeDef * Instance*
- *LTDC_InitTypeDef Init*
- *LTDC_LayerCfgTypeDef LayerCfg*
- *HAL_LockTypeDef Lock*
- *__IO HAL_LTDC_StateTypeDef State*

- `__IO uint32_t ErrorCode`

Field Documentation

- `LTDC_TypeDef* LTDC_HandleTypeDef::Instance`
LTDC Register base address
- `LTDC_InitTypeDef LTDC_HandleTypeDef::Init`
LTDC parameters
- `LTDC_LayerCfgTypeDef LTDC_HandleTypeDef::LayerCfg[MAX_LAYER]`
LTDC Layers parameters
- `HAL_LockTypeDef LTDC_HandleTypeDef::Lock`
LTDC Lock
- `__IO HAL_LTDC_StateTypeDef LTDC_HandleTypeDef::State`
LTDC state
- `__IO uint32_t LTDC_HandleTypeDef::ErrorCode`
LTDC Error code

41.2 LTDC Firmware driver API description

41.2.1 How to use this driver

1. Program the required configuration through the following parameters: the LTDC timing, the horizontal and vertical polarity, the pixel clock polarity, Data Enable polarity and the LTDC background color value using `HAL_LTDC_Init()` function
2. Program the required configuration through the following parameters: the pixel format, the blending factors, input alpha value, the window size and the image size using `HAL_LTDC_ConfigLayer()` function for foreground or/and background layer.
3. Optionally, configure and enable the CLUT using `HAL_LTDC_ConfigCLUT()` and `HAL_LTDC_EnableCLUT` functions.
4. Optionally, enable the Dither using `HAL_LTDC_EnableDither()`.
5. Optionally, configure and enable the Color keying using `HAL_LTDC_ConfigColorKeying()` and `HAL_LTDC_EnableColorKeying` functions.
6. Optionally, configure LineInterrupt using `HAL_LTDC_ProgramLineEvent()` function
7. If needed, reconfigure and change the pixel format value, the alpha value value, the window size, the window position and the layer start address for foreground or/and background layer using respectively the following functions: `HAL_LTDC_SetPixelFormat()`, `HAL_LTDC_SetAlpha()`, `HAL_LTDC_SetWindowSize()`, `HAL_LTDC_SetWindowPosition()`, `HAL_LTDC_SetAddress`.
8. Variant functions with `_NoReload` post fix allows to set the LTDC configuration/settings without immediate reload. This is useful in case when the program requires to modify serval LTDC settings (on one or both layers) then applying(reload) these settings in one shot by calling the function `HAL_LTDC_Reload`. After calling the `_NoReload` functions to set different color/format/layer settings, the program can call the function `HAL_LTDC_Reload` To apply(Reload) these settings. Function `HAL_LTDC_Reload` can be called with the parameter `ReloadType` set to `LTDC_RELOAD_IMMEDIATE` if an immediate reload is required. Function `HAL_LTDC_Reload` can be called with the parameter `ReloadType` set to `LTDC_RELOAD_VERTICAL_BLANKING` if the reload should be done in the next vertical blanking period, this option allows to avoid display flicker by applying the new settings during the vertical blanking period.
9. To control LTDC state you can use the following function: `HAL_LTDC_GetState()`

LTDC HAL driver macros list

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

- `__HAL_LTDC_ENABLE`: Enable the LTDC.

- `__HAL_LTDC_DISABLE`: Disable the LTDC.
- `__HAL_LTDC_LAYER_ENABLE`: Enable an LTDC Layer.
- `__HAL_LTDC_LAYER_DISABLE`: Disable an LTDC Layer.
- `__HAL_LTDC_RELOAD_IMMEDIATE_CONFIG`: Reload Layer Configuration.
- `__HAL_LTDC_GET_FLAG`: Get the LTDC pending flags.
- `__HAL_LTDC_CLEAR_FLAG`: Clear the LTDC pending flags.
- `__HAL_LTDC_ENABLE_IT`: Enable the specified LTDC interrupts.
- `__HAL_LTDC_DISABLE_IT`: Disable the specified LTDC interrupts.
- `__HAL_LTDC_GET_IT_SOURCE`: Check whether the specified LTDC interrupt has occurred or not.

Note: You can refer to the LTDC HAL driver header file for more useful macros

41.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC
- De-initialize the LTDC

This section contains the following APIs:

- [`HAL_LTDC_Init`](#)
- [`HAL_LTDC_DelInit`](#)
- [`HAL_LTDC_MspInit`](#)
- [`HAL_LTDC_MspDelInit`](#)
- [`HAL_LTDC_ErrorCallback`](#)
- [`HAL_LTDC_LineEventCallback`](#)
- [`HAL_LTDC_ReloadEventCallback`](#)

41.2.3 IO operation functions

This section provides function allowing to:

- Handle LTDC interrupt request

This section contains the following APIs:

- [`HAL_LTDC_IRQHandler`](#)
- [`HAL_LTDC_ErrorCallback`](#)
- [`HAL_LTDC_LineEventCallback`](#)
- [`HAL_LTDC_ReloadEventCallback`](#)

41.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the LTDC foreground or/and background parameters.
- Set the active layer.
- Configure the color keying.
- Configure the C-LUT.
- Enable / Disable the color keying.
- Enable / Disable the C-LUT.
- Update the layer position.
- Update the layer size.
- Update pixel format on the fly.
- Update transparency on the fly.
- Update address on the fly.

This section contains the following APIs:

- [*HAL_LTDC_ConfigLayer*](#)
- [*HAL_LTDC_ConfigColorKeying*](#)
- [*HAL_LTDC_ConfigCLUT*](#)
- [*HAL_LTDC_EnableColorKeying*](#)
- [*HAL_LTDC_DisableColorKeying*](#)
- [*HAL_LTDC_EnableCLUT*](#)
- [*HAL_LTDC_DisableCLUT*](#)
- [*HAL_LTDC_EnableDither*](#)
- [*HAL_LTDC_DisableDither*](#)
- [*HAL_LTDC_SetWindowSize*](#)
- [*HAL_LTDC_SetWindowPosition*](#)
- [*HAL_LTDC_SetPixelFormat*](#)
- [*HAL_LTDC_SetAlpha*](#)
- [*HAL_LTDC_SetAddress*](#)
- [*HAL_LTDC_SetPitch*](#)
- [*HAL_LTDC_ProgramLineEvent*](#)
- [*HAL_LTDC_Reload*](#)
- [*HAL_LTDC_ConfigLayer_NoReload*](#)
- [*HAL_LTDC_SetWindowSize_NoReload*](#)
- [*HAL_LTDC_SetWindowPosition_NoReload*](#)
- [*HAL_LTDC_SetPixelFormat_NoReload*](#)
- [*HAL_LTDC_SetAlpha_NoReload*](#)
- [*HAL_LTDC_SetAddress_NoReload*](#)
- [*HAL_LTDC_SetPitch_NoReload*](#)
- [*HAL_LTDC_ConfigColorKeying_NoReload*](#)
- [*HAL_LTDC_EnableColorKeying_NoReload*](#)
- [*HAL_LTDC_DisableColorKeying_NoReload*](#)
- [*HAL_LTDC_EnableCLUT_NoReload*](#)
- [*HAL_LTDC_DisableCLUT_NoReload*](#)

41.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the LTDC handle state.
- Get the LTDC handle error code.

This section contains the following APIs:

- [*HAL_LTDC_GetState*](#)
- [*HAL_LTDC_GetError*](#)

41.2.6 Detailed description of functions

HAL_LTDC_Init

Function name

HAL_StatusTypeDef HAL_LTDC_Init (LTDC_HandleTypeDef * hltdc)

Function description

Initialize the LTDC according to the specified parameters in the LTDC_InitTypeDef.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_DeInit**Function name**

HAL_StatusTypeDef HAL_LTDC_DeInit (LTDC_HandleTypeDef * hltc)

Function description

De-initialize the LTDC peripheral.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_MspInit**Function name**

void HAL_LTDC_MspInit (LTDC_HandleTypeDef * hltc)

Function description

Initialize the LTDC MSP.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_MspDeInit**Function name**

void HAL_LTDC_MspDeInit (LTDC_HandleTypeDef * hltc)

Function description

De-initialize the LTDC MSP.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_ErrorCallback**Function name**

void HAL_LTDC_ErrorCallback (LTDC_HandleTypeDef * hltc)

Function description

Error LTDC callback.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_LineEventCallback

Function name

void HAL_LTDC_LineEventCallback (LTDC_HandleTypeDef * hltc)

Function description

Line Event callback.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_ReloadEventCallback

Function name

void HAL_LTDC_ReloadEventCallback (LTDC_HandleTypeDef * hltc)

Function description

Reload Event callback.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_IRQHandler

Function name

void HAL_LTDC_IRQHandler (LTDC_HandleTypeDef * hltc)

Function description

Handle LTDC interrupt request.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_ConfigLayer

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayer (LTDC_HandleTypeDef * hltc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)

Function description

Configure the LTDC Layer according to the specified parameters in the LTDC_InitTypeDef and create the associated handle.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pLayerCfg:** pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowSize

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetWindowSize (LTDC_HandleTypeDef * hLcdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)
```

Function description

Set the LTDC window size.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **XSize:** LTDC Pixel per line
- **YSize:** LTDC Line number
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowPosition

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetWindowPosition (LTDC_HandleTypeDef * hLcdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)
```

Function description

Set the LTDC window position.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **X0:** LTDC window X offset
- **Y0:** LTDC window Y offset
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetPixelFormat

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetPixelFormat (LTDC_HandleTypeDef * hLcdc, uint32_t PixelFormat, uint32_t LayerIdx)
```

Function description

Reconfigure the pixel format.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **PixelFormat:** new pixel format value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetAlpha

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetAlpha (LTDC_HandleTypeDef * hLcdc, uint32_t Alpha, uint32_t LayerIdx)
```

Function description

Reconfigure the layer alpha value.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Alpha:** new alpha value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetAddress

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetAddress (LTDC_HandleTypeDef * hLcdc, uint32_t Address, uint32_t LayerIdx)
```

Function description

Reconfigure the frame buffer Address.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Address:** new address value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetPitch

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetPitch (LTDC_HandleTypeDef * hLcdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)
```

Function description

Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LinePitchInPixels:** New line pitch in pixels to configure for LTDC layer 'LayerIdx'.
- **LayerIdx:** LTDC layer index concerned by the modification of line pitch.

Return values

- **HAL:** status

Notes

- This function should be called only after a previous call to HAL_LTDC_ConfigLayer() to modify the default pitch configured by HAL_LTDC_ConfigLayer() when required (refer to example described just above).

HAL_LTDC_ConfigColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying (LTDC_HandleTypeDef * hLcdc, uint32_t RGBValue, uint32_t LayerIdx)

Function description

Configure the color keying.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **RGBValue:** the color key value
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_ConfigCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigCLUT (LTDC_HandleTypeDef * hLcdc, uint32_t * pCLUT, uint32_t CLUTSize, uint32_t LayerIdx)

Function description

Load the color lookup table.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pCLUT:** pointer to the color lookup table address.
- **CLUTSize:** the color lookup table size.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_EnableColorKeying (LTDC_HandleTypeDef * hLcdc, uint32_t LayerIdx)

Function description

Enable the color keying.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_DisableColorKeying (LTDC_HandleTypeDef * hLcdc, uint32_t LayerIdx)

Function description

Disable the color keying.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCLUT (LTDC_HandleTypeDef * hLcdc, uint32_t LayerIdx)

Function description

Enable the color lookup table.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCLUT (LTDC_HandleTypeDef * hLcdc, uint32_t LayerIdx)

Function description

Disable the color lookup table.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_ProgramLineEvent

Function name

`HAL_StatusTypeDef HAL_LTDC_ProgramLineEvent (LTDC_HandleTypeDef * hltc, uint32_t Line)`

Function description

Define the position of the line interrupt.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Line:** Line Interrupt Position.

Return values

- **HAL:** status

Notes

- User application may resort to `HAL_LTDC_LineEventCallback()` at line interrupt generation.

HAL_LTDC_EnableDither

Function name

`HAL_StatusTypeDef HAL_LTDC_EnableDither (LTDC_HandleTypeDef * hltc)`

Function description

Enable Dither.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_DisableDither

Function name

`HAL_StatusTypeDef HAL_LTDC_DisableDither (LTDC_HandleTypeDef * hltc)`

Function description

Disable Dither.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_Reload

Function name

`HAL_StatusTypeDef HAL_LTDC_Reload (LTDC_HandleTypeDef * hltc, uint32_t ReloadType)`

Function description

Reload LTDC Layers configuration.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

- **ReloadType:** This parameter can be one of the following values : LTDC_RELOAD_IMMEDIATE : Immediate Reload LTDC_RELOAD_VERTICAL_BLANKING : Reload in the next Vertical Blanking

Return values

- **HAL:** status

Notes

- User application may resort to HAL_LTDC_ReloadEventCallback() at reload interrupt generation.

HAL_LTDC_ConfigLayer_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_ConfigLayer_NoReload (LTDC_HandleTypeDef * hltc,  
LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)
```

Function description

Configure the LTDC Layer according to the specified without reloading parameters in the LTDC_InitTypeDef and create the associated handle.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pLayerCfg:** pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowSize_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetWindowSize_NoReload (LTDC_HandleTypeDef * hltc, uint32_t  
XSize, uint32_t YSize, uint32_t LayerIdx)
```

Function description

Set the LTDC window size without reloading.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **XSize:** LTDC Pixel per line
- **YSize:** LTDC Line number
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowPosition_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetWindowPosition_NoReload (LTDC_HandleTypeDef * hltc, uint32_t  
X0, uint32_t Y0, uint32_t LayerIdx)
```

Function description

Set the LTDC window position without reloading.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **X0:** LTDC window X offset
- **Y0:** LTDC window Y offset
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetPixelFormat_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetPixelFormat_NoReload (LTDC_HandleTypeDef * hltc, uint32_t  
PixelFormat, uint32_t LayerIdx)
```

Function description

Reconfigure the pixel format without reloading.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **PixelFormat:** new pixel format value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetAlpha_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetAlpha_NoReload (LTDC_HandleTypeDef * hltc, uint32_t Alpha,  
uint32_t LayerIdx)
```

Function description

Reconfigure the layer alpha value without reloading.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Alpha:** new alpha value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetAddress_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetAddress_NoReload (LTDC_HandleTypeDef * hltc, uint32_t Address,  
uint32_t LayerIdx)
```

Function description

Reconfigure the frame buffer Address without reloading.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Address:** new address value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetPitch_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_SetPitch_NoReload (LTDC_HandleTypeDef * hLcdc, uint32_t  
LinePitchInPixels, uint32_t LayerIdx)
```

Function description

Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LinePitchInPixels:** New line pitch in pixels to configure for LTDC layer 'LayerIdx'.
- **LayerIdx:** LTDC layer index concerned by the modification of line pitch.

Return values

- **HAL:** status

Notes

- This function should be called only after a previous call to HAL_LTDC_ConfigLayer() to modify the default pitch configured by HAL_LTDC_ConfigLayer() when required (refer to example described just above). Variant of the function HAL_LTDC_SetPitch without immediate reload.

HAL_LTDC_ConfigColorKeying_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying_NoReload (LTDC_HandleTypeDef * hLcdc, uint32_t  
RGBValue, uint32_t LayerIdx)
```

Function description

Configure the color keying without reloading.

Parameters

- **hLcdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **RGBValue:** the color key value
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableColorKeying_NoReload

Function name

```
HAL_StatusTypeDef HAL_LTDC_EnableColorKeying_NoReload (LTDC_HandleTypeDef * hLcdc, uint32_t  
LayerIdx)
```

Function description

Enable the color keying without reloading.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_DisableColorKeying_NoReload (LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)

Function description

Disable the color keying without reloading.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableCLUT_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCLUT_NoReload (LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)

Function description

Enable the color lookup table without reloading.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableCLUT_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCLUT_NoReload (LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)

Function description

Disable the color lookup table without reloading.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_GetState**Function name****HAL_LTDC_StateTypeDef HAL_LTDC_GetState (LTDC_HandleTypeDef * hltc)****Function description**

Return the LTDC handle state.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** state

HAL_LTDC_GetError**Function name****uint32_t HAL_LTDC_GetError (LTDC_HandleTypeDef * hltc)****Function description**

Return the LTDC handle error code.

Parameters

- **hltc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **LTDC:** Error Code

41.3 LTDC Firmware driver defines

41.3.1 LTDC

LTDC Alpha**LTDC_ALPHA**

LTDC Cte Alpha mask

LTDC BACK COLOR**LTDC_COLOR**

Color mask

LTDC Blending Factor1**LTDC_BLENDING_FACTOR1_CA**

Blending factor : Cte Alpha

LTDC_BLENDING_FACTOR1_PAxCA

Blending factor : Cte Alpha x Pixel Alpha

LTDC Blending Factor2

LTDC_BLENDING_FACTOR2_CA

Blending factor : Cte Alpha

LTDC_BLENDING_FACTOR2_PAxCA

Blending factor : Cte Alpha x Pixel Alpha

LTDC DE POLARITY**LTDC_DEPOLARITY_AL**

Data Enable, is active low.

LTDC_DEPOLARITY_AH

Data Enable, is active high.

LTDC Error Code**HAL_LTDC_ERROR_NONE**

LTDC No error

HAL_LTDC_ERROR_TE

LTDC Transfer error

HAL_LTDC_ERROR_FU

LTDC FIFO Underrun

HAL_LTDC_ERROR_TIMEOUT

LTDC Timeout error

LTDC Exported Macros**__HAL_LTDC_RESET_HANDLE_STATE****Description:**

- Reset LTDC handle state.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None

__HAL_LTDC_ENABLE**Description:**

- Enable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_DISABLE**Description:**

- Disable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

[__HAL_LTDC_LAYER_ENABLE](#)**Description:**

- Enable the LTDC Layer.

Parameters:

- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be enabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:

- None.

[__HAL_LTDC_LAYER_DISABLE](#)**Description:**

- Disable the LTDC Layer.

Parameters:

- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be disabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:

- None.

[__HAL_LTDC_RELOAD_IMMEDIATE_CONFIG](#)**Description:**

- Reload immediately all LTDC Layers.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

[__HAL_LTDC_VERTICAL_BLANKING_RELOAD_CONFIG](#)**Description:**

- Reload during vertical blanking period all LTDC Layers.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

[__HAL_LTDC_GET_FLAG](#)**Description:**

- Get the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
 - LTDC_FLAG_LI: Line Interrupt flag

- LTDC_FLAG_FU: FIFO Underrun Interrupt flag
- LTDC_FLAG_TE: Transfer Error interrupt flag
- LTDC_FLAG_RR: Register Reload Interrupt Flag

Return value:

- The state of FLAG (SET or RESET).

__HAL_LTDC_CLEAR_FLAG

Description:

- Clears the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
 - LTDC_FLAG_LI: Line Interrupt flag
 - LTDC_FLAG_FU: FIFO Underrun Interrupt flag
 - LTDC_FLAG_TE: Transfer Error interrupt flag
 - LTDC_FLAG_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_ENABLE_IT

Description:

- Enables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: specifies the LTDC interrupt sources to be enabled. This parameter can be any combination of the following values:
 - LTDC_IT_LI: Line Interrupt flag
 - LTDC_IT_FU: FIFO Underrun Interrupt flag
 - LTDC_IT_TE: Transfer Error interrupt flag
 - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_DISABLE_IT

Description:

- Disables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: specifies the LTDC interrupt sources to be disabled. This parameter can be any combination of the following values:
 - LTDC_IT_LI: Line Interrupt flag
 - LTDC_IT_FU: FIFO Underrun Interrupt flag
 - LTDC_IT_TE: Transfer Error interrupt flag
 - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- None

[__HAL_LTDC_GET_IT_SOURCE](#)

Description:

- Checks whether the specified LTDC interrupt has occurred or not.

Parameters:

- `__HANDLE__`: LTDC handle
- `__INTERRUPT__`: specifies the LTDC interrupt source to check. This parameter can be one of the following values:
 - `LTDC_IT_LI`: Line Interrupt flag
 - `LTDC_IT_FU`: FIFO Underrun Interrupt flag
 - `LTDC_IT_TE`: Transfer Error interrupt flag
 - `LTDC_IT_RR`: Register Reload Interrupt Flag

Return value:

- The state of INTERRUPT (SET or RESET).

[LTDC Exported Types](#)

[MAX_LAYER](#)

[LTDC Flags](#)

[LTDC_FLAG_LI](#)

[LTDC_FLAG_FU](#)

[LTDC_FLAG_TE](#)

[LTDC_FLAG_RR](#)

[LTDC HS POLARITY](#)

[LTDC_HSPOLARITY_AL](#)

Horizontal Synchronization is active low.

[LTDC_HSPOLARITY_AH](#)

Horizontal Synchronization is active high.

[LTDC Interrupts](#)

[LTDC_IT_LI](#)

[LTDC_IT_FU](#)

[LTDC_IT_TE](#)

[LTDC_IT_RR](#)

[LTDC Layer](#)

[LTDC_LAYER_1](#)

LTDC Layer 1

[LTDC_LAYER_2](#)

LTDC Layer 2

[LTDC LAYER Config](#)

LTDC_STOPPOSITION

LTDC Layer stop position

LTDC_STARTPOSITION

LTDC Layer start position

LTDC_COLOR_FRAME_BUFFER

LTDC Layer Line length

LTDC_LINE_NUMBER

LTDC Layer Line number

LTDC PC POLARITY**LTDC_PCPOLARITY_IPC**

input pixel clock.

LTDC_PCPOLARITY_IIPC

inverted input pixel clock.

LTDC Pixel format**LTDC_PIXEL_FORMAT_ARGB8888**

ARGB8888 LTDC pixel format

LTDC_PIXEL_FORMAT_RGB888

RGB888 LTDC pixel format

LTDC_PIXEL_FORMAT_RGB565

RGB565 LTDC pixel format

LTDC_PIXEL_FORMAT_ARGB1555

ARGB1555 LTDC pixel format

LTDC_PIXEL_FORMAT_ARGB4444

ARGB4444 LTDC pixel format

LTDC_PIXEL_FORMAT_L8

L8 LTDC pixel format

LTDC_PIXEL_FORMAT_AL44

AL44 LTDC pixel format

LTDC_PIXEL_FORMAT_AL88

AL88 LTDC pixel format

LTDC Reload Type**LTDC_RELOAD_IMMEDIATE**

Immediate Reload

LTDC_RELOAD_VERTICAL_BLANKING

Vertical Blanking Reload

LTDC SYNC**LTDC_HORIZONTALSYNC**

Horizontal synchronization width.

LTDC_VERTICALSYNC

Vertical synchronization height.

LTDC VS POLARITY

LTDC_VSPOLARITY_AL

Vertical Synchronization is active low.

LTDC_VSPOLARITY_AH

Vertical Synchronization is active high.

42 HAL MDIOS Generic Driver

42.1 MDIOS Firmware driver registers structures

42.1.1 MDIOS_InitTypeDef

Data Fields

- `uint32_t PortAddress`
- `uint32_t PreambleCheck`

Field Documentation

- `uint32_t MDIOS_InitTypeDef::PortAddress`

Specifies the MDIOS port address. This parameter can be a value from 0 to 31

- `uint32_t MDIOS_InitTypeDef::PreambleCheck`

Specifies whether the preamble check is enabled or disabled. This parameter can be a value of **MDIOS_Preamble Check**

42.1.2 MDIOS_HandleTypeDef

Data Fields

- `MDIOS_TypeDef * Instance`
- `MDIOS_InitTypeDef Init`
- `_IO HAL_MDIOS_StateTypeDef State`
- `HAL_LockTypeDef Lock`

Field Documentation

- `MDIOS_TypeDef* MDIOS_HandleTypeDef::Instance`

Register base address

- `MDIOS_InitTypeDef MDIOS_HandleTypeDef::Init`

MDIOS Init Structure

- `_IO HAL_MDIOS_StateTypeDef MDIOS_HandleTypeDef::State`

MDIOS communication state

- `HAL_LockTypeDef MDIOS_HandleTypeDef::Lock`

MDIOS Lock

42.2 MDIOS Firmware driver API description

42.2.1 How to use this driver

Note: A callback is executed for each generated interrupt, so the driver provide the following `HAL_MDIOS_WriteCpltCallback()`, `HAL_MDIOS_ReadCpltCallback()` and `HAL_MDIOS_ErrorCallback()`

Note: `HAL_MDIOS_IRQHandler()` must be called from the MDIOS IRQ Handler, to handle the interrupt and execute the previous callbacks (#) Reset the MDIOS peripheral and all related resources by calling the `HAL_MDIOS_DeInit()` API. (##) `HAL_MDIOS_MspDeInit()` must be implemented to reset low level resources (GPIO, Clocks, NVIC configuration ...)

42.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the MDIOS

- The following parameters can be configured:
 - Port Address
 - Preamble Check

This section contains the following APIs:

- [HAL_MDIOS_Init](#)
- [HAL_MDIOS_DelInit](#)
- [HAL_MDIOS_MsplInit](#)
- [HAL_MDIOS_MspDelInit](#)

42.2.3 IO operation functions

This section contains the following APIs:

- [HAL_MDIOS_WriteReg](#)
- [HAL_MDIOS_ReadReg](#)
- [HAL_MDIOS_GetWrittenRegAddress](#)
- [HAL_MDIOS_GetReadRegAddress](#)
- [HAL_MDIOS_ClearWriteRegAddress](#)
- [HAL_MDIOS_ClearReadRegAddress](#)
- [HAL_MDIOS_EnableEvents](#)
- [HAL_MDIOS IRQHandler](#)
- [HAL_MDIOS_WriteCpltCallback](#)
- [HAL_MDIOS_ReadCpltCallback](#)
- [HAL_MDIOS_ErrorCallback](#)
- [HAL_MDIOS_WakeUpCallback](#)

42.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the MDIOS.

- HAL_MDIOS_GetState() API, helpful to check in run-time the state.
- HAL_MDIOS_GetError() API, returns the errors occurred during data transfer.

This section contains the following APIs:

- [HAL_MDIOS_GetError](#)
- [HAL_MDIOS_GetState](#)

42.2.5 Detailed description of functions

HAL_MDIOS_Init

Function name

`HAL_StatusTypeDef HAL_MDIOS_Init (MDIOS_HandleTypeDef * hmdios)`

Function description

Initializes the MDIOS according to the specified parameters in the MDIOS_InitTypeDef and creates the associated handle .

Parameters

- **hmdios:** pointer to a MDIOS_HandleTypeDef structure that contains the configuration information for MDIOS module

Return values

- **HAL:** status

HAL_MDIOS_DelInit**Function name**

```
HAL_StatusTypeDef HAL_MDIOS_DelInit (MDIOS_HandleTypeDef * hmdios)
```

Function description

DeInitializes the MDIOS peripheral.

Parameters

- **hmdios:** MDIOS handle

Return values

- **HAL:** status

HAL_MDIOS_MspInit**Function name**

```
void HAL_MDIOS_MspInit (MDIOS_HandleTypeDef * hmdios)
```

Function description

MDIOS MSP Init.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_MspDelInit**Function name**

```
void HAL_MDIOS_MspDelInit (MDIOS_HandleTypeDef * hmdios)
```

Function description

MDIOS MSP Delinit.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_WriteReg**Function name**

```
HAL_StatusTypeDef HAL_MDIOS_WriteReg (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum,  
uint16_t Data)
```

Function description

Writes to an MDIOS output register.

Parameters

- **hmdios:** mdios handle
- **RegNum:** MDIOS output register address

- **Data:** Data to write

Return values

- **HAL:** status

HAL_MDIOS_ReadReg

Function name

```
HAL_StatusTypeDef HAL_MDIOS_ReadReg (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum,  
uint16_t * pData)
```

Function description

Reads an MDIOS input register.

Parameters

- **hmdios:** mdios handle
- **RegNum:** MDIOS input register address
- **pData:** pointer to Data

Return values

- **HAL:** status

HAL_MDIOS_GetWrittenRegAddress

Function name

```
uint32_t HAL_MDIOS_GetWrittenRegAddress (MDIOS_HandleTypeDef * hmdios)
```

Function description

Gets Written registers by MDIO master.

Parameters

- **hmdios:** mdios handle

Return values

- **bit:** map of written registers addresses

HAL_MDIOS_GetReadRegAddress

Function name

```
uint32_t HAL_MDIOS_GetReadRegAddress (MDIOS_HandleTypeDef * hmdios)
```

Function description

Gets Read registers by MDIO master.

Parameters

- **hmdios:** mdios handle

Return values

- **bit:** map of read registers addresses

HAL_MDIOS_ClearWriteRegAddress

Function name

```
HAL_StatusTypeDef HAL_MDIOS_ClearWriteRegAddress (MDIOS_HandleTypeDef * hmdios, uint32_t  
RegNum)
```

Function description

Clears Write registers flag.

Parameters

- **hmdios:** mdios handle
- **RegNum:** registers addresses to be cleared

Return values

- **HAL:** status

HAL_MDIOS_ClearReadRegAddress

Function name

HAL_StatusTypeDef HAL_MDIOS_ClearReadRegAddress (MDIOS_HandleTypeDef * hmdios, uint32_t RegNum)

Function description

Clears Read register flag.

Parameters

- **hmdios:** mdios handle
- **RegNum:** registers addresses to be cleared

Return values

- **HAL:** status

HAL_MDIOS_EnableEvents

Function name

HAL_StatusTypeDef HAL_MDIOS_EnableEvents (MDIOS_HandleTypeDef * hmdios)

Function description

Enables Events for MDIOS peripheral.

Parameters

- **hmdios:** mdios handle

Return values

- **HAL:** status

HAL_MDIOS_IRQHandler

Function name

void HAL_MDIOS_IRQHandler (MDIOS_HandleTypeDef * hmdios)

Function description

This function handles MDIOS interrupt request.

Parameters

- **hmdios:** MDIOS handle

Return values

- **None:**

HAL_MDIOS_WriteCpltCallback

Function name

```
void HAL_MDIOS_WriteCpltCallback (MDIOS_HandleTypeDef * hmdios)
```

Function description

Write Complete Callback.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_ReadCpltCallback

Function name

```
void HAL_MDIOS_ReadCpltCallback (MDIOS_HandleTypeDef * hmdios)
```

Function description

Read Complete Callback.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_ErrorCallback

Function name

```
void HAL_MDIOS_ErrorCallback (MDIOS_HandleTypeDef * hmdios)
```

Function description

Error Callback.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_WakeUpCallback

Function name

```
void HAL_MDIOS_WakeUpCallback (MDIOS_HandleTypeDef * hmdios)
```

Function description

MDIOS WAKEUP interrupt callback.

Parameters

- **hmdios:** mdios handle

Return values

- **None:**

HAL_MDIOS_GetError

Function name

uint32_t HAL_MDIOS_GetError (MDIOS_HandleTypeDef * hmdios)

Function description

Gets MDIOS error flags.

Parameters

- **hmdios:** mdios handle

Return values

- **bit:** map of occurred errors

HAL_MDIOS_GetState

Function name

HAL_MDIOS_StateTypeDef HAL_MDIOS_GetState (MDIOS_HandleTypeDef * hmdios)

Function description

Return the MDIOS HAL state.

Parameters

- **hmdios:** mdios handle

Return values

- **HAL:** state

42.3 MDIOS Firmware driver defines

42.3.1 MDIOS

MDIOS Exported Macros

__HAL_MDIOS_RESET_HANDLE_STATE

Description:

- Reset MDIOS handle state.

Parameters:

- __HANDLE__: MDIOS handle.

Return value:

- None

__HAL_MDIOS_ENABLE

Description:

- Enable/Disable the MDIOS peripheral.

Parameters:

- __HANDLE__: specifies the MDIOS handle.

Return value:

- None

__HAL_MDIOS_DISABLE

__HAL_MDIOS_ENABLE_IT

Description:

- Enable the MDIOS device interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: specifies the MDIOS interrupt sources to be enabled. This parameter can be one or a combination of the following values:
 - MDIOS_IT_WRITE: Register write interrupt
 - MDIOS_IT_READ: Register read interrupt
 - MDIOS_IT_ERROR: Error interrupt

Return value:

- None

__HAL_MDIOS_DISABLE_IT

Description:

- Disable the MDIOS device interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: specifies the MDIOS interrupt sources to be disabled. This parameter can be one or a combination of the following values:
 - MDIOS_IT_WRITE: Register write interrupt
 - MDIOS_IT_READ: Register read interrupt
 - MDIOS_IT_ERROR: Error interrupt

Return value:

- None

__HAL_MDIOS_GET_WRITE_FLAG

Description:

- Set MDIOS slave get write register flag.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: specifies the write register flag

Return value:

- The: state of write flag

__HAL_MDIOS_GET_READ_FLAG

Description:

- MDIOS slave get read register flag.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: specifies the read register flag

Return value:

- The: state of read flag

__HAL_MDIOS_GET_ERROR_FLAG

Description:

- MDIOS slave get interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: specifies the Error flag. This parameter can be one or a combination of the following values:
 - MDIOS_TURNAROUND_ERROR_FLAG: Register write interrupt
 - MDIOS_START_ERROR_FLAG: Register read interrupt
 - MDIOS_PREAMBLE_ERROR_FLAG: Error interrupt

Return value:

- The: state of the error flag

[__HAL_MDIOS_CLEAR_ERROR_FLAG](#)**Description:**

- MDIOS slave clear interrupt.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __FLAG__: specifies the Error flag. This parameter can be one or a combination of the following values:
 - MDIOS_TURNAROUND_ERROR_FLAG: Register write interrupt
 - MDIOS_START_ERROR_FLAG: Register read interrupt
 - MDIOS_PREAMBLE_ERROR_FLAG: Error interrupt

Return value:

- none

[__HAL_MDIOS_GET_IT_SOURCE](#)**Description:**

- Checks whether the specified MDIOS interrupt is set or not.

Parameters:

- __HANDLE__: specifies the MDIOS handle.
- __INTERRUPT__: specifies the MDIOS interrupt sources This parameter can be one or a combination of the following values:
 - MDIOS_IT_WRITE: Register write interrupt
 - MDIOS_IT_READ: Register read interrupt
 - MDIOS_IT_ERROR: Error interrupt

Return value:

- The: state of the interrupt source

[__HAL_MDIOS_WAKEUP_EXTI_ENABLE_IT](#)**Description:**

- Enable the MDIOS WAKEUP Exti Line.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be enabled. This parameter can be:
 - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None.

[__HAL_MDIOS_WAKEUP_EXTI_GET_FLAG](#)**Description:**

- checks whether the specified MDIOS WAKEUP Exti interrupt flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be cleared. This parameter can be:
 - MDIOS_WAKEUP_EXTI_LINE

Return value:

- EXTI: MDIOS WAKEUP Line Status.

[__HAL_MDIOS_WAKEUP_EXTI_CLEAR_FLAG](#)**Description:**

- Clear the MDIOS WAKEUP Exti flag.

Parameters:

- __EXTI_LINE__: specifies the MDIOS WAKEUP Exti sources to be cleared. This parameter can be:
 - MDIOS_WAKEUP_EXTI_LINE

Return value:

- None.

[__HAL_MDIOS_WAKEUP_EXTI_ENABLE_RISING_EDGE](#)**Description:**

- enable rising edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled. This parameter can be:
 - ETH_WAKEUP_EXTI_LINE

Return value:

- None

[__HAL_MDIOS_WAKEUP_EXTI_ENABLE_FALLING_EDGE](#)**Description:**

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled. This parameter can be:
 - ETH_WAKEUP_EXTI_LINE

Return value:

- None

[__HAL_MDIOS_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE](#)**Description:**

- enable falling edge interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__: specifies the ETH WAKEUP EXTI sources to be disabled. This parameter can be:
 - ETH_WAKEUP_EXTI_LINE

Return value:

- None

[__HAL_MDIOS_WAKEUP_EXTI_GENERATE_SWIT](#)**Description:**

- Generates a Software interrupt on selected EXTI line.

Parameters:

- `__EXTI_LINE__`: specifies the MDIOS WAKEUP EXTI sources to be disabled. This parameter can be:
 - `MDIOS_WAKEUP_EXTI_LINE`

Return value:

- None

MDIOS Input Output Registers Definitions`MDIOS_REG0``MDIOS_REG1``MDIOS_REG2``MDIOS_REG3``MDIOS_REG4``MDIOS_REG5``MDIOS_REG6``MDIOS_REG7``MDIOS_REG8``MDIOS_REG9``MDIOS_REG10``MDIOS_REG11``MDIOS_REG12``MDIOS_REG13``MDIOS_REG14``MDIOS_REG15``MDIOS_REG16``MDIOS_REG17``MDIOS_REG18``MDIOS_REG19``MDIOS_REG20``MDIOS_REG21``MDIOS_REG22``MDIOS_REG23`

MDIOS_REG24

MDIOS_REG25

MDIOS_REG26

MDIOS_REG27

MDIOS_REG28

MDIOS_REG29

MDIOS_REG30

MDIOS_REG31

MDIOS Interrupt Flags

MDIOS_TURNAROUND_ERROR_FLAG

MDIOS_START_ERROR_FLAG

MDIOS_PREAMBLE_ERROR_FLAG

Interrupt Sources

MDIOS_IT_WRITE

MDIOS_IT_READ

MDIOS_IT_ERROR

MDIOS Preamble Check

MDIOS_PREAMBLE_CHECK_ENABLE

MDIOS_PREAMBLE_CHECK_DISABLE

MDIOS Registers Flags

MDIOS_REG0_FLAG

MDIOS_REG1_FLAG

MDIOS_REG2_FLAG

MDIOS_REG3_FLAG

MDIOS_REG4_FLAG

MDIOS_REG5_FLAG

MDIOS_REG6_FLAG

MDIOS_REG7_FLAG

MDIOS_REG8_FLAG

MDIOS_REG9_FLAG

MDIOS_REG10_FLAG

MDIOS_REG11_FLAG

MDIOS_REG12_FLAG

MDIOS_REG13_FLAG

MDIOS_REG14_FLAG

MDIOS_REG15_FLAG

MDIOS_REG16_FLAG

MDIOS_REG17_FLAG

MDIOS_REG18_FLAG

MDIOS_REG19_FLAG

MDIOS_REG20_FLAG

MDIOS_REG21_FLAG

MDIOS_REG22_FLAG

MDIOS_REG23_FLAG

MDIOS_REG24_FLAG

MDIOS_REG25_FLAG

MDIOS_REG26_FLAG

MDIOS_REG27_FLAG

MDIOS_REG28_FLAG

MDIOS_REG29_FLAG

MDIOS_REG30_FLAG

MDIOS_REG31_FLAG

MDIOS_ALLREG_FLAG

MDIOS Wakeup Line

MDIOS_WAKEUP_EXTI_LINE

43 HAL MDMA Generic Driver

43.1 MDMA Firmware driver registers structures

43.1.1 MDMA_InitTypeDef

Data Fields

- `uint32_t Request`
- `uint32_t TransferTriggerMode`
- `uint32_t Priority`
- `uint32_t Endianness`
- `uint32_t SourceInc`
- `uint32_t DestinationInc`
- `uint32_t SourceDataSize`
- `uint32_t DestDataSize`
- `uint32_t DataAlignment`
- `uint32_t BufferTransferLength`
- `uint32_t SourceBurst`
- `uint32_t DestBurst`
- `int32_t SourceBlockAddressOffset`
- `int32_t DestBlockAddressOffset`

Field Documentation

- `uint32_t MDMA_InitTypeDef::Request`

Specifies the MDMA request. This parameter can be a value of **MDMA Request selection**

- `uint32_t MDMA_InitTypeDef::TransferTriggerMode`

Specifies the Trigger Transfer mode : each request triggers a : a buffer transfer, a block transfer, a repeated block transfer or a linked list transfer This parameter can be a value of **MDMA Transfer Trigger Mode**

- `uint32_t MDMA_InitTypeDef::Priority`

Specifies the software priority for the MDMAy channelx. This parameter can be a value of **MDMA Priority level**

- `uint32_t MDMA_InitTypeDef::Endianness`

Specifies if the MDMA transactions preserve the Little endianness. This parameter can be a value of **MDMA Endianness**

- `uint32_t MDMA_InitTypeDef::SourceInc`

Specifies if the Source increment mode . This parameter can be a value of **MDMA Source increment mode**

- `uint32_t MDMA_InitTypeDef::DestinationInc`

Specifies if the Destination increment mode . This parameter can be a value of **MDMA Destination increment mode**

- `uint32_t MDMA_InitTypeDef::SourceDataSize`

Specifies the source data size. This parameter can be a value of **MDMA Source data size**

- `uint32_t MDMA_InitTypeDef::DestDataSize`

Specifies the destination data size. This parameter can be a value of **MDMA Destination data size**

- `uint32_t MDMA_InitTypeDef::DataAlignment`

Specifies the source to destination Memory data packing/padding mode. This parameter can be a value of **MDMA data alignment**

- **`uint32_t MDMA_InitTypeDef::BufferTransferLength`**

Specifies the buffer Transfer Length (number of bytes), this is the number of bytes to be transferred in a single transfer (1 byte to 128 bytes)

- **`uint32_t MDMA_InitTypeDef::SourceBurst`**

Specifies the Burst transfer configuration for the source memory transfers. It specifies the amount of data to be transferred in a single non interruptable transaction. This parameter can be a value of **MDMA Source burst**

Note:

- : the burst may be FIXED/INCR based on SourceInc value , the BURST must be programmed as to ensure that the burst size will be lower than than BufferTransferLength

- **`uint32_t MDMA_InitTypeDef::DestBurst`**

Specifies the Burst transfer configuration for the destination memory transfers. It specifies the amount of data to be transferred in a single non interruptable transaction. This parameter can be a value of **MDMA Destination burst**

Note:

- : the burst may be FIXED/INCR based on DestinationInc value , the BURST must be programmed as to ensure that the burst size will be lower than than BufferTransferLength

- **`int32_t MDMA_InitTypeDef::SourceBlockAddressOffset`**

this field specifies the Next block source address offset signed value : if > 0 then increment the next block source Address by offset from where the last block ends if < 0 then decrement the next block source Address by offset from where the last block ends if == 0, the next block source address starts from where the last block ends

- **`int32_t MDMA_InitTypeDef::DestBlockAddressOffset`**

this field specifies the Next block destination address offset signed value : if > 0 then increment the next block destination Address by offset from where the last block ends if < 0 then decrement the next block destination Address by offset from where the last block ends if == 0, the next block destination address starts from where the last block ends

43.1.2 MDMA_LinkNodeTypeDef

Data Fields

- **`_IO uint32_t CTCR`**
- **`_IO uint32_t CBNDTR`**
- **`_IO uint32_t CSAR`**
- **`_IO uint32_t CDAR`**
- **`_IO uint32_t CBRUR`**
- **`_IO uint32_t CLAR`**
- **`_IO uint32_t CTBR`**
- **`_IO uint32_t Reserved`**
- **`_IO uint32_t CMAR`**
- **`_IO uint32_t CMDR`**

Field Documentation

- **`_IO uint32_t MDMA_LinkNodeTypeDef::CTCR`**

New CTCR register configuration for the given MDMA linked list node

- **`_IO uint32_t MDMA_LinkNodeTypeDef::CBNDTR`**

New CBNDTR register configuration for the given MDMA linked list node

- `__IO uint32_t MDMA_LinkNodeTypeDef::CSAR`
New CSAR register configuration for the given MDMA linked list node
- `__IO uint32_t MDMA_LinkNodeTypeDef::CDAR`
New CDAR register configuration for the given MDMA linked list node
- `__IO uint32_t MDMA_LinkNodeTypeDef::CBRUR`
New CBRUR register configuration for the given MDMA linked list node
- `__IO uint32_t MDMA_LinkNodeTypeDef::CLAR`
New CLAR register configuration for the given MDMA linked list node
- `__IO uint32_t MDMA_LinkNodeTypeDef::CTBR`
New CTBR register configuration for the given MDMA linked list node
- `__IO uint32_t MDMA_LinkNodeTypeDef::Reserved`
Reserved register
- `__IO uint32_t MDMA_LinkNodeTypeDef::CMAR`
New CMAR register configuration for the given MDMA linked list node
- `__IO uint32_t MDMA_LinkNodeTypeDef::CMDR`
New CMDR register configuration for the given MDMA linked list node

43.1.3 MDMA_LinkNodeConfTypeDef

Data Fields

- `MDMA_InitTypeDef Init`
- `uint32_t SrcAddress`
- `uint32_t DstAddress`
- `uint32_t BlockDataLength`
- `uint32_t BlockCount`
- `uint32_t PostRequestMaskAddress`
- `uint32_t PostRequestMaskData`

Field Documentation

- `MDMA_InitTypeDef MDMA_LinkNodeConfTypeDef::Init`
configuration of the specified MDMA Linked List Node
- `uint32_t MDMA_LinkNodeConfTypeDef::SrcAddress`
The source memory address for the Linked list Node
- `uint32_t MDMA_LinkNodeConfTypeDef::DstAddress`
The destination memory address for the Linked list Node
- `uint32_t MDMA_LinkNodeConfTypeDef::BlockDataLength`
The length of a block transfer in bytes
- `uint32_t MDMA_LinkNodeConfTypeDef::BlockCount`
The number of a blocks to be transfer
- `uint32_t MDMA_LinkNodeConfTypeDef::PostRequestMaskAddress`
specifies the address to be updated (written) with PostRequestMaskData after a request is served.
PostRequestMaskAddress and PostRequestMaskData could be used to automatically clear a peripheral flag
when the request is served
- `uint32_t MDMA_LinkNodeConfTypeDef::PostRequestMaskData`

specifies the value to be written to PostRequestMaskAddress after a request is served.
PostRequestMaskAddress and PostRequestMaskData could be used to automatically clear a peripheral flag when the request is served

43.1.4 `__MDMA_HandleTypeDef`

Data Fields

- `MDMA_Channel_TypeDef * Instance`
- `MDMA_InitTypeDef Init`
- `HAL_LockTypeDef Lock`
- `__IO HAL_MDMA_StateTypeDef State`
- `void * Parent`
- `void(* XferCpltCallback`
- `void(* XferBufferCpltCallback`
- `void(* XferBlockCpltCallback`
- `void(* XferRepeatBlockCpltCallback`
- `void(* XferErrorCallback`
- `void(* XferAbortCallback`
- `MDMA_LinkNodeTypeDef * FirstLinkedListNodeAddress`
- `MDMA_LinkNodeTypeDef * LastLinkedListNodeAddress`
- `uint32_t LinkedListNodeCounter`
- `__IO uint32_t ErrorCode`

Field Documentation

- `MDMA_Channel_TypeDef* __MDMA_HandleTypeDef::Instance`
Register base address
- `MDMA_InitTypeDef __MDMA_HandleTypeDef::Init`
MDMA communication parameters
- `HAL_LockTypeDef __MDMA_HandleTypeDef::Lock`
MDMA locking object
- `__IO HAL_MDMA_StateTypeDef __MDMA_HandleTypeDef::State`
MDMA transfer state
- `void* __MDMA_HandleTypeDef::Parent`
Parent object state
- `void(* __MDMA_HandleTypeDef::XferCpltCallback)(struct __MDMA_HandleTypeDef *hmdma)`
MDMA transfer complete callback
- `void(* __MDMA_HandleTypeDef::XferBufferCpltCallback)(struct __MDMA_HandleTypeDef *hmdma)`
MDMA buffer transfer complete callback
- `void(* __MDMA_HandleTypeDef::XferBlockCpltCallback)(struct __MDMA_HandleTypeDef *hmdma)`
MDMA block transfer complete callback
- `void(* __MDMA_HandleTypeDef::XferRepeatBlockCpltCallback)(struct __MDMA_HandleTypeDef *hmdma)`
MDMA block transfer repeat callback
- `void(* __MDMA_HandleTypeDef::XferErrorCallback)(struct __MDMA_HandleTypeDef *hmdma)`
MDMA transfer error callback
- `void(* __MDMA_HandleTypeDef::XferAbortCallback)(struct __MDMA_HandleTypeDef *hmdma)`

- MDMA transfer Abort callback
- **`MDMA_LinkNodeTypeDef* __MDMA_HandleTypeDef::FirstLinkedListNodeAddress`**
specifies the first node address of the transfer list (after the initial node defined by the Init struct) this parameter is used internally by the MDMA driver to construct the linked list node
- **`MDMA_LinkNodeTypeDef* __MDMA_HandleTypeDef::LastLinkedListNodeAddress`**
specifies the last node address of the transfer list this parameter is used internally by the MDMA driver to construct the linked list node
- **`uint32_t __MDMA_HandleTypeDef::LinkedListNodeCounter`**
Number of nodes in the MDMA linked list
- **`_IO uint32_t __MDMA_HandleTypeDef::ErrorCode`**
MDMA Error code

43.2 MDMA Firmware driver API description

43.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the MDMA Channel (except for internal SRAM/FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and MDMA requests.
2. For a given Channel use HAL_MDMA_Init function to program the required configuration through the following parameters: transfer request , channel priority, data endianness, Source increment, destination increment , source data size, destination data size, data alignment, source Burst, destination Burst , buffer Transfer Length, Transfer Trigger Mode (buffer transfer, block transfer, repeated block transfer or full transfer) source and destination block address offset, mask address and data. If using the MDMA in linked list mode then use function HAL_MDMA_LinkedList_CreateNode to fill a transfer node. Note that parameters given to the function HAL_MDMA_Init corresponds always to the node zero. Use function HAL_MDMA_LinkedList_AddNode to connect the created node to the linked list at a given position. User can make a linked list circular using function HAL_MDMA_LinkedList_EnableCircularMode , this function will automatically connect the last node of the list to the first one in order to make the list circular. In this case the linked list will loop on node 1 : first node connected after the initial transfer defined by the HAL_MDMA_Init

Note:

The initial transfer itself (node 0 corresponding to the Init). User can disable the circular mode using function HAL_MDMA_LinkedList_DisableCircularMode, this function will then remove the connection between last node and first one. Function HAL_MDMA_LinkedList_RemoveNode can be used to remove (disconnect) a node from the transfer linked list. When a linked list is circular (last node connected to first one), if removing node1 (node where the linked list loops), the linked list remains circular and node 2 becomes the first one. Note that if the linked list is made circular the transfer will loop infinitely (or until aborted by the user).

- User can select the transfer trigger mode (parameter TransferTriggerMode) to define the amount of data to be transfer upon a request :
 - MDMA_BUFFER_TRANSFER : each request triggers a transfer of BufferTransferLength data with BufferTransferLength defined within the HAL_MDMA_Init.
 - MDMA_BLOCK_TRANSFER : each request triggers a transfer of a block with block size defined within the function HAL_MDMA_Start/HAL_MDMA_Start_IT or within the current linked list node parameters.
 - MDMA_REPEAT_BLOCK_TRANSFER : each request triggers a transfer of a number of blocks with block size and number of blocks defined within the function HAL_MDMA_Start/HAL_MDMA_Start_IT or within the current linked list node parameters.
 - MDMA_FULL_TRANSFER : each request triggers a full transfer all blocks and all nodes(if a linked list has been created using HAL_MDMA_LinkedList_CreateNode \ HAL_MDMA_LinkedList_AddNode).

Polling mode IO operation

- Use HAL_MDMA_Start() to start MDMA transfer after the configuration of Source address and destination address and the Length of data to be transferred.

- Use HAL_MDMA_PollForTransfer() to poll for the end of current transfer or a transfer level In this case a fixed Timeout can be configured by User depending from his application.
- Use HAL_MDMA_Abort() function to abort the current transfer : blocking method this API returns when the abort ends or timeout (should not be called from an interrupt service routine).

Interrupt mode IO operation

- Configure the MDMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the MDMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_MDMA_Start_IT() to start MDMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the MDMA interrupt is configured.
- Use HAL_MDMA_IRQHandler() called under MDMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_MDMA_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of MDMA handle structure).
- Use HAL_MDMA_Abort_IT() function to abort the current transfer : non-blocking method. This API returns immediately then the callback XferAbortCallback (if specified by the user) is asserted once the MDMA channel has effectively aborted. (could be called from an interrupt service routine).
- Use functions HAL_MDMA_RegisterCallback and HAL_MDMA_UnRegisterCallback respectively to register/unregister user callbacks from the following list :
 - XferCpltCallback : transfer complete callback.
 - XferBufferCpltCallback : buffer transfer complete callback.
 - XferBlockCpltCallback : block transfer complete callback.
 - XferRepeatBlockCpltCallback : repeated block transfer complete callback.
 - XferErrorCallback : transfer error callback.
 - XferAbortCallback : transfer abort complete callback.
- If the transfer Request corresponds to SW request (MDMA_REQUEST_SW) User can use function HAL_MDMA_GenerateSWRequest to trigger requests manually. Function HAL_MDMA_GenerateSWRequest must be used with the following precautions:
 - This function returns an error if used while the Transfer has not started.
 - If used while the current request has not been served yet (current request transfer on going) this function returns an error and the new request is ignored. Generally this function should be used in conjunctions with the MDMA callbacks:
 - example 1:
 - Configure a transfer with request set to MDMA_REQUEST_SW and trigger mode set to MDMA_BUFFER_TRANSFER
 - Register a callback for buffer transfer complete (using callback ID set to HAL_MDMA_XFER_BUFFERCPLT_CB_ID)
 - After calling HAL_MDMA_Start_IT the MDMA will issue the transfer of a first BufferTransferLength data.
 - When the buffer transfer complete callback is asserted first buffer has been transferred and user can ask for a new buffer transfer request using HAL_MDMA_GenerateSWRequest.
 - example 2:
 - Configure a transfer with request set to MDMA_REQUEST_SW and trigger mode set to MDMA_BLOCK_TRANSFER
 - Register a callback for block transfer complete (using callback ID HAL_MDMA_XFER_BLOCKCPLT_CB_ID)
 - After calling HAL_MDMA_Start_IT the MDMA will issue the transfer of a first block of data.
 - When the block transfer complete callback is asserted the first block has been transferred and user can ask for a new block transfer request using HAL_MDMA_GenerateSWRequest.

Use HAL_MDMA_GetState() function to return the MDMA state and HAL_MDMA_GetError() in case of error detection.

MDMA HAL driver macros list

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

- `_HAL_MDMA_ENABLE`: Enable the specified MDMA Stream.
- `_HAL_MDMA_DISABLE`: Disable the specified MDMA Stream.
- `_HAL_MDMA_GET_FLAG`: Get the MDMA Stream pending flags.
- `_HAL_MDMA_CLEAR_FLAG`: Clear the MDMA Stream pending flags.
- `_HAL_MDMA_ENABLE_IT`: Enable the specified MDMA Stream interrupts.
- `_HAL_MDMA_DISABLE_IT`: Disable the specified MDMA Stream interrupts.
- `_HAL_MDMA_GET_IT_SOURCE`: Check whether the specified MDMA Stream interrupt has occurred or not.

Note: You can refer to the header file of the MDMA HAL driver for more useful macros.

43.2.2 Initialization and de-initialization functions

This section provides functions allowing to : Initialize and de-initialize the MDMA channel. Register and Unregister MDMA callbacks

The `HAL_MDMA_Init()` function follows the MDMA channel configuration procedures as described in reference manual. The `HAL_MDMA_DelInit` function allows to deinitialize the MDMA channel.
`HAL_MDMA_RegisterCallback` and `HAL_MDMA_UnRegisterCallback` functions allows respectively to register/unregister an MDMA callback function.

This section contains the following APIs:

- [`HAL_MDMA_Init`](#)
- [`HAL_MDMA_DelInit`](#)
- [`HAL_MDMA_ConfigPostRequestMask`](#)
- [`HAL_MDMA_RegisterCallback`](#)
- [`HAL_MDMA_UnRegisterCallback`](#)

43.2.3 Linked list operation functions

This section provides functions allowing to:

- Create a linked list node
- Add a node to the MDMA linked list
- Remove a node from the MDMA linked list
- Enable/Disable linked list circular mode

This section contains the following APIs:

- [`HAL_MDMA_LinkedList_CreateNode`](#)
- [`HAL_MDMA_LinkedList_AddNode`](#)
- [`HAL_MDMA_LinkedList_RemoveNode`](#)
- [`HAL_MDMA_LinkedList_EnableCircularMode`](#)
- [`HAL_MDMA_LinkedList_DisableCircularMode`](#)

43.2.4 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start MDMA transfer
- Configure the source, destination address and data length and Start MDMA transfer with interrupt
- Abort MDMA transfer
- Poll for transfer complete

- Generate a SW request (when Request is set to MDMA_REQUEST_SW)
- Handle MDMA interrupt request

This section contains the following APIs:

- [**HAL_MDMA_Start**](#)
- [**HAL_MDMA_Start_IT**](#)
- [**HAL_MDMA_Abort**](#)
- [**HAL_MDMA_Abort_IT**](#)
- [**HAL_MDMA_PollForTransfer**](#)
- [**HAL_MDMA_GenerateSWRequest**](#)
- [**HAL_MDMA_IRQHandler**](#)

43.2.5 State and Errors functions

This subsection provides functions allowing to

- Check the MDMA state
- Get error code

This section contains the following APIs:

- [**HAL_MDMA_GetState**](#)
- [**HAL_MDMA_GetError**](#)

43.2.6 Detailed description of functions

HAL_MDMA_Init

Function name

HAL_StatusTypeDef HAL_MDMA_Init (MDMA_HandleTypeDef * hmdma)

Function description

Initializes the MDMA according to the specified parameters in the MDMA_InitTypeDef and create the associated handle.

Parameters

- **hmdma:** Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.

Return values

- **HAL:** status

HAL_MDMA_DelInit

Function name

HAL_StatusTypeDef HAL_MDMA_DelInit (MDMA_HandleTypeDef * hmdma)

Function description

Deinitializes the MDMA peripheral.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.

Return values

- **HAL:** status

HAL_MDMA_ConfigPostRequestMask

Function name

```
HAL_StatusTypeDef HAL_MDMA_ConfigPostRequestMask (MDMA_HandleTypeDef * hmdma, uint32_t MaskAddress, uint32_t MaskData)
```

Function description

Config the Post request Mask address and Mask data.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **MaskAddress:** specifies the address to be updated (written) with MaskData after a request is served.
- **MaskData:** specifies the value to be written to MaskAddress after a request is served. MaskAddress and MaskData could be used to automatically clear a peripheral flag when the request is served.

Return values

- **HAL:** status

HAL_MDMA_RegisterCallback

Function name

```
HAL_StatusTypeDef HAL_MDMA_RegisterCallback (MDMA_HandleTypeDef * hmdma, HAL_MDMA_CallbackIDTypeDef CallbackID, void(*)(MDMA_HandleTypeDef * _hmdma) pCallback)
```

Function description

Register callbacks.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **CallbackID:** User Callback identifier
- **pCallback:** pointer to callback function.

Return values

- **HAL:** status

HAL_MDMA_UnRegisterCallback

Function name

```
HAL_StatusTypeDef HAL_MDMA_UnRegisterCallback (MDMA_HandleTypeDef * hmdma, HAL_MDMA_CallbackIDTypeDef CallbackID)
```

Function description

UnRegister callbacks.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **CallbackID:** User Callback identifier a HAL_MDMA_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status

HAL_MDMA_LinkedList_CreateNode

Function name

```
HAL_StatusTypeDef HAL_MDMA_LinkedList_CreateNode (MDMA_LinkNodeTypeDef * pNode,  
MDMA_LinkNodeConfTypeDef * pNodeConfig)
```

Function description

Initializes an MDMA Link Node according to the specified parameters in the pMDMA_LinkedListNodeConfig .

Parameters

- **pNode:** Pointer to a MDMA_LinkNodeTypeDef structure that contains Linked list node registers configurations.
- **pNodeConfig:** Pointer to a MDMA_LinkNodeConfTypeDef structure that contains the configuration information for the specified MDMA Linked List Node.

Return values

- **HAL:** status

HAL_MDMA_LinkedList_AddNode

Function name

```
HAL_StatusTypeDef HAL_MDMA_LinkedList_AddNode (MDMA_HandleTypeDef * hmdma,  
MDMA_LinkNodeTypeDef * pNewNode, MDMA_LinkNodeTypeDef * pPrevNode)
```

Function description

Connect a node to the linked list.

Parameters

- **hmdma:** : Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **pNewNode:** : Pointer to a MDMA_LinkNodeTypeDef structure that contains Linked list node to be add to the list.
- **pPrevNode:** : Pointer to the new node position in the linked list or zero to insert the new node at the end of the list

Return values

- **HAL:** status

HAL_MDMA_LinkedList_RemoveNode

Function name

```
HAL_StatusTypeDef HAL_MDMA_LinkedList_RemoveNode (MDMA_HandleTypeDef * hmdma,  
MDMA_LinkNodeTypeDef * pNode)
```

Function description

Disconnect/Remove a node from the transfer linked list.

Parameters

- **hmdma:** : Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **pNode:** : Pointer to a MDMA_LinkNodeTypeDef structure that contains Linked list node to be removed from the list.

Return values

- **HAL:** status

HAL_MDMA_LinkedList_EnableCircularMode

Function name

HAL_StatusTypeDef HAL_MDMA_LinkedList_EnableCircularMode (MDMA_HandleTypeDef * hmdma)

Function description

Make the linked list circular by connecting the last node to the first.

Parameters

- **hmdma:** : Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

HAL_MDMA_LinkedList_DisableCircularMode

Function name

HAL_StatusTypeDef HAL_MDMA_LinkedList_DisableCircularMode (MDMA_HandleTypeDef * hmdma)

Function description

Disable the linked list circular mode by setting the last node connection to null.

Parameters

- **hmdma:** : Pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

HAL_MDMA_Start

Function name

HAL_StatusTypeDef HAL_MDMA_Start (MDMA_HandleTypeDef * hmdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t BlockDataLength, uint32_t BlockCount)

Function description

Starts the MDMA Transfer.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.
- **SrcAddress:** : The source memory Buffer address
- **DstAddress:** : The destination memory Buffer address
- **BlockDataLength:** : The length of a block transfer in bytes
- **BlockCount:** : The number of a blocks to be transfer

Return values

- **HAL:** status

HAL_MDMA_Start_IT

Function name

HAL_StatusTypeDef HAL_MDMA_Start_IT (MDMA_HandleTypeDef * hmdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t BlockDataLength, uint32_t BlockCount)

Function description

Starts the MDMA Transfer with interrupts enabled.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.
- **SrcAddress:** : The source memory Buffer address
- **DstAddress:** : The destination memory Buffer address
- **BlockDataLength:** : The length of a block transfer in bytes
- **BlockCount:** : The number of a blocks to be transfer

Return values

- **HAL:** status

HAL_MDMA_Abort

Function name

HAL_StatusTypeDef HAL_MDMA_Abort (MDMA_HandleTypeDef * hmdma)

Function description

Aborts the MDMA Transfer.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

Notes

- After disabling a MDMA Stream, a check for wait until the MDMA Channel is effectively disabled is added. If a Stream is disabled while a data transfer is ongoing, the current data will be transferred and the Stream will be effectively disabled only after the transfer of this single data is finished.

HAL_MDMA_Abort_IT

Function name

HAL_StatusTypeDef HAL_MDMA_Abort_IT (MDMA_HandleTypeDef * hmdma)

Function description

Aborts the MDMA Transfer in Interrupt mode.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.

Return values

- **HAL:** status

HAL_MDMA_PollForTransfer

Function name

HAL_StatusTypeDef HAL_MDMA_PollForTransfer (MDMA_HandleTypeDef * hmdma, uint32_t CompleteLevel, uint32_t Timeout)

Function description

Polling for transfer complete.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Channel.
- **CompleteLevel:** Specifies the MDMA level complete.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_MDMA_GenerateSWRequest

Function name

HAL_StatusTypeDef HAL_MDMA_GenerateSWRequest (MDMA_HandleTypeDef * hmdma)

Function description

Generate an MDMA SW request trigger to activate the request on the given Channel.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.

Return values

- **HAL:** status

HAL_MDMA_IRQHandler

Function name

void HAL_MDMA_IRQHandler (MDMA_HandleTypeDef * hmdma)

Function description

Handles MDMA interrupt request.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.

Return values

- **None:**

HAL_MDMA_GetState

Function name

HAL_MDMA_StateTypeDef HAL_MDMA_GetState (MDMA_HandleTypeDef * hmdma)

Function description

Returns the MDMA state.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.

Return values

- **HAL:** state

HAL_MDMA_GetError

Function name

uint32_t HAL_MDMA_GetError (MDMA_HandleTypeDef * hmdma)

Function description

Return the MDMA error code.

Parameters

- **hmdma:** : pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified MDMA Stream.

Return values

- **MDMA:** Error Code

43.3 MDMA Firmware driver defines

43.3.1 MDMA

MDMA data alignment

MDMA_DATAALIGN_PACKENABLE

The source data is packed/un-packed into the destination data size All data are right aligned, in Little Endian mode.

MDMA_DATAALIGN_RIGHT

Right Aligned, padded w/ 0s (default)

MDMA_DATAALIGN_RIGHT_SIGNED

Right Aligned, Sign extended , Note : this mode is allowed only if the Source data size smaller than Destination data size

MDMA_DATAALIGN_LEFT

Left Aligned (padded with 0s)

MDMA Destination burst

MDMA_DEST_BURST_SINGLE

single transfer

MDMA_DEST_BURST_2BEATS

Burst 2 beats

MDMA_DEST_BURST_4BEATS

Burst 4 beats

MDMA_DEST_BURST_8BEATS

Burst 8 beats

MDMA_DEST_BURST_16BEATS

Burst 16 beats

MDMA_DEST_BURST_32BEATS

Burst 32 beats

MDMA_DEST_BURST_64BEATS

Burst 64 beats

MDMA_DEST_BURST_128BEATS

Burst 128 beats

MDMA Destination data size**MDMA_DEST_DATASIZE_BYTE**

Destination data size is Byte

MDMA_DEST_DATASIZE_HALFWORD

Destination data size is half word

MDMA_DEST_DATASIZE_WORD

Destination data size is word

MDMA_DEST_DATASIZE_DOUBLEWORD

Destination data size is double word

MDMA Destination increment mode**MDMA_DEST_INC_DISABLE**

Source address pointer is fixed

MDMA_DEST_INC_BYTE

Source address pointer is incremented by a BYTE (8 bits)

MDMA_DEST_INC_HALFWORD

Source address pointer is incremented by a half Word (16 bits)

MDMA_DEST_INC_WORD

Source address pointer is incremented by a Word (32 bits)

MDMA_DEST_INC_DOUBLEWORD

Source address pointer is incremented by a double Word (64 bits))

MDMA_DEST_DEC_BYTE

Source address pointer is decremented by a BYTE (8 bits)

MDMA_DEST_DEC_HALFWORD

Source address pointer is decremented by a half Word (16 bits)

MDMA_DEST_DEC_WORD

Source address pointer is decremented by a Word (32 bits)

MDMA_DEST_DEC_DOUBLEWORD

Source address pointer is decremented by a double Word (64 bits))

MDMA Endianness**MDMA_LITTLE_ENDIANNESS_PRESERVE**

little endianness preserve

MDMA_LITTLE_BYTE_ENDIANNESS_EXCHANGE

BYTEs endianness exchange when destination data size is> Byte

MDMA_LITTLE_HALFWORD_ENDIANNESS_EXCHANGE

HALF WORDs endianness exchange when destination data size is > HALF WORD

MDMA_LITTLE_WORD_ENDIANNESS_EXCHANGE

WORDS endianness exchange when destination data size is > DOUBLE WORD

MDMA Error Codes**HAL_MDMA_ERROR_NONE**

No error

HAL_MDMA_ERROR_READ_XFER

Read Transfer error

HAL_MDMA_ERROR_WRITE_XFER

Write Transfer error

HAL_MDMA_ERROR_MASK_DATA

Error Mask Data error

HAL_MDMA_ERROR_LINKED_LIST

Linked list Data error

HAL_MDMA_ERROR_ALIGNMENT

Address/Size alignment error

HAL_MDMA_ERROR_BLOCK_SIZE

Block Size error

HAL_MDMA_ERROR_TIMEOUT

Timeout error

HAL_MDMA_ERROR_NO_XFER

Abort or SW trigger requested with no Xfer ongoing

HAL_MDMA_ERROR_BUSY

DeInit or SW trigger requested with Xfer ongoing

MDMA Exported Macros**__HAL_MDMA_ENABLE****Description:**

- Enable the specified MDMA Channel.

Parameters:

- __HANDLE__: MDMA handle

Return value:

- None

__HAL_MDMA_DISABLE**Description:**

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: MDMA handle

Return value:

- None

[__HAL_MDMA_GET_FLAG](#)**Description:**

- Get the MDMA Channel pending flags.

Parameters:

- __HANDLE__: MDMA handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
 - MDMA_FLAG_TE : Transfer Error flag.
 - MDMA_FLAG_CTC : Channel Transfer Complete flag.
 - MDMA_FLAG_BRT : Block Repeat Transfer flag.
 - MDMA_FLAG_BT : Block Transfer complete flag.
 - MDMA_FLAG_BFTC : BuFfer Transfer Complete flag.
 - MDMA_FLAG_CRQA : Channel ReQest Active flag.

Return value:

- The: state of FLAG (SET or RESET).

[__HAL_MDMA_CLEAR_FLAG](#)**Description:**

- Clear the MDMA Stream pending flags.

Parameters:

- __HANDLE__: MDMA handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
 - MDMA_FLAG_TE : Transfer Error flag.
 - MDMA_FLAG_CTC : Channel Transfer Complete flag.
 - MDMA_FLAG_BRT : Block Repeat Transfer flag.
 - MDMA_FLAG_BT : Block Transfer complete flag.
 - MDMA_FLAG_BFTC : BuFfer Transfer Complete flag.

Return value:

- None

[__HAL_MDMA_ENABLE_IT](#)**Description:**

- Enables the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: MDMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
 - MDMA_IT_TE : Transfer Error interrupt mask
 - MDMA_IT_CTC : Channel Transfer Complete interrupt mask
 - MDMA_IT_BRT : Block Repeat Transfer interrupt mask
 - MDMA_IT_BT : Block Transfer interrupt mask
 - MDMA_IT_BFTC : BuFfer Transfer Complete interrupt mask

Return value:

- None

__HAL_MDMA_DISABLE_IT

Description:

- Disables the specified MDMA Channel interrupts.

Parameters:

- __HANDLE__: MDMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
 - MDMA_IT_TE : Transfer Error interrupt mask
 - MDMA_IT_CTC : Channel Transfer Complete interrupt mask
 - MDMA_IT_BRT : Block Repeat Transfer interrupt mask
 - MDMA_IT_BT : Block Transfer interrupt mask
 - MDMA_IT_BFTC : BuFfer Transfer Complete interrupt mask

Return value:

- None

__HAL_MDMA_GET_IT_SOURCE

Description:

- Checks whether the specified MDMA Channel interrupt is enabled or not.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt source to check.
 - MDMA_IT_TE : Transfer Error interrupt mask
 - MDMA_IT_CTC : Channel Transfer Complete interrupt mask
 - MDMA_IT_BRT : Block Repeat Transfer interrupt mask
 - MDMA_IT_BT : Block Transfer interrupt mask
 - MDMA_IT_BFTC : BuFfer Transfer Complete interrupt mask

Return value:

- The: state of MDMA_IT (SET or RESET).

MDMA flag definitions

MDMA_FLAG_TE

Transfer Error flag

MDMA_FLAG_CTC

Channel Transfer Complete flag

MDMA_FLAG_BRT

Block Repeat Transfer complete flag

MDMA_FLAG_BT

Block Transfer complete flag

MDMA_FLAG_BFTC

BuFfer Transfer complete flag

MDMA_FLAG_CRQA

Channel ReQuest Active flag

MDMA interrupt enable definitions

MDMA_IT_TE

Transfer Error interrupt

MDMA_IT_CTC

Channel Transfer Complete interrupt

MDMA_IT_BRT

Block Repeat Transfer interrupt

MDMA_IT_BT

Block Transfer interrupt

MDMA_IT_BFTC

Buffer Transfer Complete interrupt

MDMA Priority level**MDMA_PRIORITY_LOW**

Priority level: Low

MDMA_PRIORITY_MEDIUM

Priority level: Medium

MDMA_PRIORITY_HIGH

Priority level: High

MDMA_PRIORITY VERY_HIGH

Priority level: Very High

MDMA Request selection**MDMA_REQUEST_DMA1_Stream0_TC**

MDMA HW request is DMA1 Stream 0 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream1_TC

MDMA HW request is DMA1 Stream 1 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream2_TC

MDMA HW request is DMA1 Stream 2 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream3_TC

MDMA HW request is DMA1 Stream 3 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream4_TC

MDMA HW request is DMA1 Stream 4 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream5_TC

MDMA HW request is DMA1 Stream 5 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream6_TC

MDMA HW request is DMA1 Stream 6 Transfer Complete Flag

MDMA_REQUEST_DMA1_Stream7_TC

MDMA HW request is DMA1 Stream 7 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream0_TC

MDMA HW request is DMA2 Stream 0 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream1_TC

MDMA HW request is DMA2 Stream 1 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream2_TC

MDMA HW request is DMA2 Stream 2 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream3_TC

MDMA HW request is DMA2 Stream 3 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream4_TC

MDMA HW request is DMA2 Stream 4 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream5_TC

MDMA HW request is DMA2 Stream 5 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream6_TC

MDMA HW request is DMA2 Stream 6 Transfer Complete Flag

MDMA_REQUEST_DMA2_Stream7_TC

MDMA HW request is DMA2 Stream 7 Transfer Complete Flag

MDMA_REQUEST_LTDC_LINE_IT

MDMA HW request is LTDC Line interrupt Flag

MDMA_REQUEST_JPEG_INFIFO_TH

MDMA HW request is JPEG Input FIFO threshold Flag

MDMA_REQUEST_JPEG_INFIFO_NF

MDMA HW request is JPEG Input FIFO not full Flag

MDMA_REQUEST_JPEG_OUTFIFO_TH

MDMA HW request is JPEG Output FIFO threshold Flag

MDMA_REQUEST_JPEG_OUTFIFO_NE

MDMA HW request is JPEG Output FIFO not empty Flag

MDMA_REQUEST_JPEG_END_CONVERSION

MDMA HW request is JPEG End of conversion Flag

MDMA_REQUEST_QUADSPI_FIFO_TH

MDMA HW request is QSPI FIFO threshold Flag

MDMA_REQUEST_QUADSPI_TC

MDMA HW request is QSPI Transfer complete Flag

MDMA_REQUEST_DMA2D_CLUT_TC

MDMA HW request is DMA2D CLUT Transfer Complete Flag

MDMA_REQUEST_DMA2D_TC

MDMA HW request is DMA2D Transfer Complete Flag

MDMA_REQUEST_DMA2D_TW

MDMA HW request is DMA2D Transfer Watermark Flag

MDMA_REQUEST_SDMMC1_END_DATA

MDMA HW request is SDMMC1 End of Data Flag

MDMA_REQUEST_SW

MDMA SW request

MDMA Source burst**MDMA_SOURCE_BURST_SINGLE**

single transfer

MDMA_SOURCE_BURST_2BEATS

Burst 2 beats

MDMA_SOURCE_BURST_4BEATS

Burst 4 beats

MDMA_SOURCE_BURST_8BEATS

Burst 8 beats

MDMA_SOURCE_BURST_16BEATS

Burst 16 beats

MDMA_SOURCE_BURST_32BEATS

Burst 32 beats

MDMA_SOURCE_BURST_64BEATS

Burst 64 beats

MDMA_SOURCE_BURST_128BEATS

Burst 128 beats

MDMA Source data size**MDMA_SRC_DATASIZE_BYTE**

Source data size is Byte

MDMA_SRC_DATASIZE_HALFWORD

Source data size is half word

MDMA_SRC_DATASIZE_WORD

Source data size is word

MDMA_SRC_DATASIZE_DOUBLEWORD

Source data size is double word

MDMA Source increment mode**MDMA_SRC_INC_DISABLE**

Source address pointer is fixed

MDMA_SRC_INC_BYTE

Source address pointer is incremented by a BYTE (8 bits)

MDMA_SRC_INC_HALFWORD

Source address pointer is incremented by a half Word (16 bits)

MDMA_SRC_INC_WORD

Source address pointer is incremented by a Word (32 bits)

MDMA_SRC_INC_DOUBLEWORD

Source address pointer is incremented by a double Word (64 bits))

MDMA_SRC_DEC_BYTE

Source address pointer is decremented by a BYTE (8 bits)

MDMA_SRC_DEC_HALFWORD

Source address pointer is decremented by a half Word (16 bits)

MDMA_SRC_DEC_WORD

Source address pointer is decremented by a Word (32 bits)

MDMA_SRC_DEC_DOUBLEWORD

Source address pointer is decremented by a double Word (64 bits))

MDMA Transfer Trigger Mode**MDMA_BUFFER_TRANSFER**

Each MDMA request (SW or HW) triggers a buffer transfer

MDMA_BLOCK_TRANSFER

Each MDMA request (SW or HW) triggers a block transfer

MDMA_REPEAT_BLOCK_TRANSFER

Each MDMA request (SW or HW) triggers a repeated block transfer

MDMA_FULL_TRANSFER

Each MDMA request (SW or HW) triggers a Full transfer or a linked list transfer if any

44 HAL MMC Generic Driver

44.1 MMC Firmware driver registers structures

44.1.1 HAL_MMC_CardInfoTypeDef

Data Fields

- `uint32_t CardType`
- `uint32_t Class`
- `uint32_t RelCardAdd`
- `uint32_t BlockNbr`
- `uint32_t BlockSize`
- `uint32_t LogBlockNbr`
- `uint32_t LogBlockSize`

Field Documentation

- `uint32_t HAL_MMC_CardInfoTypeDef::CardType`
Specifies the card Type
- `uint32_t HAL_MMC_CardInfoTypeDef::Class`
Specifies the class of the card class
- `uint32_t HAL_MMC_CardInfoTypeDef::RelCardAdd`
Specifies the Relative Card Address
- `uint32_t HAL_MMC_CardInfoTypeDef::BlockNbr`
Specifies the Card Capacity in blocks
- `uint32_t HAL_MMC_CardInfoTypeDef::BlockSize`
Specifies one block size in bytes
- `uint32_t HAL_MMC_CardInfoTypeDef::LogBlockNbr`
Specifies the Card logical Capacity in blocks
- `uint32_t HAL_MMC_CardInfoTypeDef::LogBlockSize`
Specifies logical block size in bytes

44.1.2 MMC_HandleTypeDef

Data Fields

- `MMC_TypeDef * Instance`
- `MMC_InitTypeDef Init`
- `HAL_LockTypeDef Lock`
- `uint32_t * pTxBuffPtr`
- `uint32_t TxXferSize`
- `uint32_t * pRxBuffPtr`
- `uint32_t RxXferSize`
- `_IO uint32_t Context`
- `_IO HAL_MMC_StateTypeDef State`
- `_IO uint32_t ErrorCode`

- ***HAL_MMC_CardInfoTypeDef MmcCard***
- ***uint32_t CSD***
- ***uint32_t CID***

Field Documentation

- ***MMC_TypeDef* MMC_HandleTypeDef::Instance***
MMC registers base address
- ***MMC_InitTypeDef MMC_HandleTypeDef::Init***
MMC required parameters
- ***HAL_LockTypeDef MMC_HandleTypeDef::Lock***
MMC locking object
- ***uint32_t* MMC_HandleTypeDef::pTxBuffPtr***
Pointer to MMC Tx transfer Buffer
- ***uint32_t MMC_HandleTypeDef::TxXferSize***
MMC Tx Transfer size
- ***uint32_t* MMC_HandleTypeDef::pRxBuffPtr***
Pointer to MMC Rx transfer Buffer
- ***uint32_t MMC_HandleTypeDef::RxXferSize***
MMC Rx Transfer size
- ***_IO uint32_t MMC_HandleTypeDef::Context***
MMC transfer context
- ***_IO HAL_MMC_StateTypeDef MMC_HandleTypeDef::State***
MMC card State
- ***_IO uint32_t MMC_HandleTypeDef::ErrorCode***
MMC Card Error codes
- ***HAL_MMC_CardInfoTypeDef MMC_HandleTypeDef::MmcCard***
MMC Card information
- ***uint32_t MMC_HandleTypeDef::CSD[4]***
MMC card specific data table
- ***uint32_t MMC_HandleTypeDef::CID[4]***
MMC card identification number table

44.1.3 HAL_MMC_CardCSDTypeDef

Data Fields

- ***_IO uint8_t CSDStruct***
- ***_IO uint8_t SysSpecVersion***
- ***_IO uint8_t Reserved1***
- ***_IO uint8_t TAAC***
- ***_IO uint8_t NSAC***
- ***_IO uint8_t MaxBusClkFrec***
- ***_IO uint16_t CardComdClasses***
- ***_IO uint8_t RdBlockLen***
- ***_IO uint8_t PartBlockRead***

- `__IO uint8_t WrBlockMisalign`
- `__IO uint8_t RdBlockMisalign`
- `__IO uint8_t DSRImpl`
- `__IO uint8_t Reserved2`
- `__IO uint32_t DeviceSize`
- `__IO uint8_t MaxRdCurrentVDDMin`
- `__IO uint8_t MaxRdCurrentVDDMax`
- `__IO uint8_t MaxWrCurrentVDDMin`
- `__IO uint8_t MaxWrCurrentVDDMax`
- `__IO uint8_t DeviceSizeMul`
- `__IO uint8_t EraseGrSize`
- `__IO uint8_t EraseGrMul`
- `__IO uint8_t WrProtectGrSize`
- `__IO uint8_t WrProtectGrEnable`
- `__IO uint8_t ManDeflECC`
- `__IO uint8_t WrSpeedFact`
- `__IO uint8_t MaxWrBlockLen`
- `__IO uint8_t WriteBlockPaPartial`
- `__IO uint8_t Reserved3`
- `__IO uint8_t ContentProtectAppli`
- `__IO uint8_t FileFormatGrouop`
- `__IO uint8_t CopyFlag`
- `__IO uint8_t PermWrProtect`
- `__IO uint8_t TempWrProtect`
- `__IO uint8_t FileFormat`
- `__IO uint8_t ECC`
- `__IO uint8_t CSD_CRC`
- `__IO uint8_t Reserved4`

Field Documentation

- `__IO uint8_t HAL_MMC_CardCSDTypeDef::CSDStruct`
CSD structure
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::SysSpecVersion`
System specification version
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved1`
Reserved
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::TAAC`
Data read access time 1
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::NSAC`
Data read access time 2 in CLK cycles
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxBusClkFrec`
Max. bus clock frequency
- `__IO uint16_t HAL_MMC_CardCSDTypeDef::CardComdClasses`
Card command classes
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockLen`
Max. read data block length

- `__IO uint8_t HAL_MMC_CardCSDTypeDef::PartBlockRead`
Partial blocks for read allowed
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::WrBlockMisalign`
Write block misalignment
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockMisalign`
Read block misalignment
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::DSRImpl`
DSR implemented
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved2`
Reserved
- `__IO uint32_t HAL_MMC_CardCSDTypeDef::DeviceSize`
Device Size
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMin`
Max. read current @ VDD min
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMax`
Max. read current @ VDD max
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMin`
Max. write current @ VDD min
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMax`
Max. write current @ VDD max
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::DeviceSizeMul`
Device size multiplier
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrSize`
Erase group size
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrMul`
Erase group size multiplier
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrSize`
Write protect group size
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrEnable`
Write protect group enable
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::ManDefIECC`
Manufacturer default ECC
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::WrSpeedFact`
Write speed factor
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrBlockLen`
Max. write data block length
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::WriteBlockPaPartial`
Partial blocks for write allowed
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved3`
Reserved
- `__IO uint8_t HAL_MMC_CardCSDTypeDef::ContentProtectAppli`

- Content protection application
- `__IO uint8_t HAL_MMCCardCSDTypeDef::FileFormatGroup`
 - File format group
 - `__IO uint8_t HAL_MMCCardCSDTypeDef::CopyFlag`
 - Copy flag (OTP)
 - `__IO uint8_t HAL_MMCCardCSDTypeDef::PermWrProtect`
 - Permanent write protection
 - `__IO uint8_t HAL_MMCCardCSDTypeDef::TempWrProtect`
 - Temporary write protection
 - `__IO uint8_t HAL_MMCCardCSDTypeDef::FileFormat`
 - File format
 - `__IO uint8_t HAL_MMCCardCSDTypeDef::ECC`
 - ECC code
 - `__IO uint8_t HAL_MMCCardCSDTypeDef::CSD_CRC`
 - CSD CRC
 - `__IO uint8_t HAL_MMCCardCSDTypeDef::Reserved4`
 - Always 1

44.1.4 HAL_MMCCardCIDTypedef

Data Fields

- `__IO uint8_t ManufacturerID`
- `__IO uint16_t OEM_AppID`
- `__IO uint32_t ProdName1`
- `__IO uint8_t ProdName2`
- `__IO uint8_t ProdRev`
- `__IO uint32_t ProdSN`
- `__IO uint8_t Reserved1`
- `__IO uint16_t ManufactDate`
- `__IO uint8_t CID_CRC`
- `__IO uint8_t Reserved2`

Field Documentation

- `__IO uint8_t HAL_MMCCardCIDTypedef::ManufacturerID`
 - Manufacturer ID
- `__IO uint16_t HAL_MMCCardCIDTypedef::OEM_AppID`
 - OEM/Application ID
- `__IO uint32_t HAL_MMCCardCIDTypedef::ProdName1`
 - Product Name part1
- `__IO uint8_t HAL_MMCCardCIDTypedef::ProdName2`
 - Product Name part2
- `__IO uint8_t HAL_MMCCardCIDTypedef::ProdRev`
 - Product Revision
- `__IO uint32_t HAL_MMCCardCIDTypedef::ProdSN`

- Product Serial Number
- `__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved1`
Reserved1
- `__IO uint16_t HAL_MMC_CardCIDTypeDef::ManufactureDate`
Manufacturing Date
- `__IO uint8_t HAL_MMC_CardCIDTypeDef::CID_CRC`
CID CRC
- `__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved2`
Always 1

44.2 MMC Firmware driver API description

44.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in `HAL_MMC_MspInit()` function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with MMC and eMMC cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implement the `HAL_MMC_MspInit()` API:
 - a. Enable the SDMMC interface clock using `__HAL_RCC_SDMMC_CLK_ENABLE();`
 - b. SDMMC pins configuration for MMC card
 - Enable the clock for the SDMMC GPIOs using the functions `__HAL_RCC_GPIOx_CLK_ENABLE();`
 - Configure these SDMMC pins as alternate function pull-up using `HAL_GPIO_Init()` and according to your pin assignment;
 - c. NVIC configuration if you need to use interrupt process when using DMA transfer.
 - Configure the SDMMC interrupt priorities using functions `HAL_NVIC_SetPriority();`
 - Enable the NVIC SDMMC IRQs using function `HAL_NVIC_EnableIRQ()`
 - SDMMC interrupts are managed using the macros `__HAL_MMC_ENABLE_IT()` and `__HAL_MMC_DISABLE_IT()` inside the communication process.
 - SDMMC interrupts pending bits are managed using the macros `__HAL_MMC_GET_IT()` and `__HAL_MMC_CLEAR_IT()`
 - d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC IP are used.
2. At this stage, you can perform MMC read/write/erase operations after MMC card initialization

MMC Card Initialization and configuration

To initialize the MMC Card, use the `HAL_MMC_Init()` function. It initializes SDMMC IP (STM32 side) and the MMC Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Initialize the SDMMC peripheral interface with default configuration. The initialization process is done at 400KHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. The MMC Card frequency (SDMMC_CK) is computed as follows: $SDMMC_CK = SDMMCCLK / (2 * ClockDiv)$ In initialization mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through `SDMMC_Init()` and `SDMMC_PowerState_ON()` SDMMC low level APIs.
2. Initialize the MMC card. The API used is `HAL_MMC_InitCard()`. This phase allows the card initialization and identification and check the MMC Card type (Standard Capacity or High Capacity) The initialization flow is

compatible with MMC standard. This API (HAL_MMC_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.

3. Configure the MMC Card Data transfer frequency. By Default, the card transfer frequency by adjusting the "ClockDiv" field. In transfer mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding MMC Card according to the address read with the step 2.
5. Configure the MMC Card in wide bus mode: 4-bits data.

MMC Card Read operation

- You can read from MMC card in polling mode by using function HAL_MMC_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can read from MMC card in DMA mode by using function HAL_MMC_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.

MMC Card Write operation

- You can write to MMC card in polling mode by using function HAL_MMC_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.
- You can write to MMC card in DMA mode by using function HAL_MMC_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.

MMC card CID register

- The HAL_MMC_GetCardCID() API allows to get the parameters of the CID register. Some of the CID parameters are useful for card initialization and identification.

MMC HAL driver macros list

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

- __HAL_MMC_ENABLE_IT: Enable the MMC device interrupt
- __HAL_MMC_DISABLE_IT: Disable the MMC device interrupt
- __HAL_MMC_GET_FLAG:Check whether the specified MMC flag is set or not
- __HAL_MMC_CLEAR_FLAG: Clear the MMC's pending flags

Note:

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

44.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the MMC card device to be ready for use.

This section contains the following APIs:

- [HAL_MMC_Init](#)
- [HAL_MMC_InitCard](#)
- [HAL_MMC_DelInit](#)
- [HAL_MMC_MspInit](#)
- [HAL_MMC_MspDelInit](#)

44.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to MMC card.

This section contains the following APIs:

- [*HAL_MMC_ReadBlocks*](#)
- [*HAL_MMC_WriteBlocks*](#)
- [*HAL_MMC_ReadBlocks_IT*](#)
- [*HAL_MMC_WriteBlocks_IT*](#)
- [*HAL_MMC_ReadBlocks_DMA*](#)
- [*HAL_MMC_WriteBlocks_DMA*](#)
- [*HAL_MMC_Erase*](#)
- [*HAL_MMC_IRQHandler*](#)
- [*HAL_MMC_GetState*](#)
- [*HAL_MMC_GetError*](#)
- [*HAL_MMC_TxCpltCallback*](#)
- [*HAL_MMC_RxCpltCallback*](#)
- [*HAL_MMC_ErrorCallback*](#)
- [*HAL_MMC_AbortCallback*](#)

44.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the MMC card operations and get the related information

This section contains the following APIs:

- [*HAL_MMC_GetCardCID*](#)
- [*HAL_MMC_GetCardCSD*](#)
- [*HAL_MMC_GetCardInfo*](#)
- [*HAL_MMC_ConfigWideBusOperation*](#)
- [*HAL_MMC_GetCardState*](#)
- [*HAL_MMC_Abort*](#)
- [*HAL_MMC_Abort_IT*](#)

44.2.5 Detailed description of functions

HAL_MMC_Init

Function name

HAL_StatusTypeDef HAL_MMC_Init (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC according to the specified parameters in the MMC_HandleTypeDef and create the associated handle.

Parameters

- **hmmc:** Pointer to the MMC handle

Return values

- **HAL:** status

HAL_MMC_InitCard

Function name

HAL_StatusTypeDef HAL_MMC_InitCard (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC Card.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

Notes

- This function initializes the MMC card. It could be used when a card re-initialization is needed.

HAL_MMC_DelInit**Function name**

HAL_StatusTypeDef HAL_MMC_DelInit (MMC_HandleTypeDef * hmmc)

Function description

De-Initializes the MMC card.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

HAL_MMC_MspInit**Function name**

void HAL_MMC_MspInit (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC MSP.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_MspDelInit**Function name**

void HAL_MMC_MspDelInit (MMC_HandleTypeDef * hmmc)

Function description

De-Initialize MMC MSP.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_ReadBlocks**Function name**

HAL_StatusTypeDef HAL_MMC_ReadBlocks (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of MMC blocks to read
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Allows to write block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of MMC blocks to write
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_Erase

Function name

HAL_StatusTypeDef HAL_MMC_Erase (MMC_HandleTypeDef * hmmc, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Erases the specified memory area of the given MMC card.

Parameters

- **hmmc:** Pointer to MMC handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_IT (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the IT transfer process through the MMC Rx interrupt event.

HAL_MMC_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_IT (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the IT transfer process through the MMC Tx interrupt event.

HAL_MMC_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_DMA (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the DMA transfer process through the MMC Rx interrupt event.

HAL_MMC_WriteBlocks_DMA

Function name

```
HAL_StatusTypeDef HAL_MMC_WriteBlocks_DMA (MMC_HandleTypeDef * hmmc, uint8_t * pData,  
uint32_t BlockAdd, uint32_t NumberOfBlocks)
```

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the DMA transfer process through the MMC Tx interrupt event.

HAL_MMC_IRQHandler

Function name

```
void HAL_MMC_IRQHandler (MMC_HandleTypeDef * hmmc)
```

Function description

This function handles MMC card interrupt request.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_TxCpltCallback

Function name

```
void HAL_MMC_TxCpltCallback (MMC_HandleTypeDef * hmmc)
```

Function description

Tx Transfer completed callbacks.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_RxCpltCallback

Function name

void HAL_MMC_RxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Rx Transfer completed callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_ErrorCallback

Function name

void HAL_MMC_ErrorCallback (MMC_HandleTypeDef * hmmc)

Function description

MMC error callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_AbortCallback

Function name

void HAL_MMC_AbortCallback (MMC_HandleTypeDef * hmmc)

Function description

MMC Abort callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigWideBusOperation (MMC_HandleTypeDef * hmmc, uint32_t WideMode)

Function description

Enables wide bus operation for the requested card if supported by card.

Parameters

- **hmmc:** Pointer to MMC handle
- **WideMode:** Specifies the MMC card wide bus mode This parameter can be one of the following values:
 - SDMMC_BUS_WIDE_8B: 8-bit data transfer
 - SDMMC_BUS_WIDE_4B: 4-bit data transfer
 - SDMMC_BUS_WIDE_1B: 1-bit data transfer

Return values

- **HAL:** status

HAL_MMC_GetCardState

Function name

HAL_MMC_CardStateTypedef HAL_MMC_GetCardState (MMC_HandleTypeDef * hmmc)

Function description

Gets the current mmc card data state.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **Card:** state

HAL_MMC_GetCardCID

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCID (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCIDTypeDef * pCID)

Function description

Returns information the information of the card which are stored on the CID register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCID:** Pointer to a HAL_MMC_CIDTypeDef structure that contains all CID register parameters

Return values

- **HAL:** status

HAL_MMC_GetCardCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCSD (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCSDTypeDef * pCSD)

Function description

Returns information the information of the card which are stored on the CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCSD:** Pointer to a HAL_MMC_CardInfoTypeDef structure that contains all CSD register parameters

Return values

- **HAL:** status

HAL_MMC_GetCardInfo

Function name

```
HAL_StatusTypeDef HAL_MMC_GetCardInfo (MMC_HandleTypeDef * hmmc,  
HAL_MMC_CardInfoTypeDef * pCardInfo)
```

Function description

Gets the MMC card info.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCardInfo:** Pointer to the HAL_MMC_CardInfoTypeDef structure that will contain the MMC card status information

Return values

- **HAL:** status

HAL_MMC_GetState

Function name

```
HAL_MMC_StateTypeDef HAL_MMC_GetState (MMC_HandleTypeDef * hmmc)
```

Function description

return the MMC state

Parameters

- **hmmc:** Pointer to mmc handle

Return values

- **HAL:** state

HAL_MMC_GetError

Function name

```
uint32_t HAL_MMC_GetError (MMC_HandleTypeDef * hmmc)
```

Function description

Return the MMC error code.

Parameters

- **hmmc:** : pointer to a MMC_HandleTypeDef structure that contains the configuration information.

Return values

- **MMC:** Error Code

HAL_MMC_Abort

Function name

```
HAL_StatusTypeDef HAL_MMC_Abort (MMC_HandleTypeDef * hmmc)
```

Function description

Abort the current transfer and disable the MMC.

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure that contains the configuration information for MMC module.

Return values

- **HAL:** status

HAL_MMC_Abort_IT

Function name

HAL_StatusTypeDef HAL_MMC_Abort_IT (MMC_HandleTypeDef * hmmc)

Function description

Abort the current transfer and disable the MMC (IT mode).

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure that contains the configuration information for MMC module.

Return values

- **HAL:** status

44.3 MMC Firmware driver defines

44.3.1 MMC

MMC Error status enumeration Structure definition

HAL_MMC_ERROR_NONE

No error

HAL_MMC_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_MMC_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_MMC_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_MMC_ERROR_DATA_TIMEOUT

Data timeout

HAL_MMC_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_MMC_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_MMC_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_MMC_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_MMC_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_MMC_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_MMC_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_MMC_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_MMC_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_MMC_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_MMC_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_MMC_ERROR_CC_ERR

Internal card controller error

HAL_MMC_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_MMC_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_MMC_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_MMC_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_MMC_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_MMC_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_MMC_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_MMC_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_MMC_ERROR_INVALID_VOLTRANGE

Error in case of invalid voltage range

HAL_MMC_ERROR_ADDR_OUT_OF_RANGE

Error when addressed block is out of range

HAL_MMC_ERROR_REQUEST_NOT_APPLICABLE

Error when command request is not applicable

HAL_MMC_ERROR_PARAM

the used parameter is not valid

HAL_MMC_ERROR_UNSUPPORTED_FEATURE

Error when feature is not insupported

HAL_MMC_ERROR_BUSY

Error when transfer process is busy

HAL_MMC_ERROR_DMA

Error while DMA transfer

HAL_MMC_ERROR_TIMEOUT

Timeout error

MMC context enumeration**MMC_CONTEXT_NONE**

None

MMC_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

MMC_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

MMC_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

MMC_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

MMC_CONTEXT_IT

Process in Interrupt mode

MMC_CONTEXT_DMA

Process in DMA mode

MMC Voltage mode**MMC_HIGH_VOLTAGE_RANGE**

VALUE OF ARGUMENT

MMC_DUAL_VOLTAGE_RANGE

VALUE OF ARGUMENT

eMMC_HIGH_VOLTAGE_RANGE

for eMMC> 2Gb sector mode

eMMC_DUAL_VOLTAGE_RANGE

for eMMC> 2Gb sector mode

MMC_INVALID_VOLTAGE_RANGE***MMC Memory Cards*****MMC_LOW_CAPACITY_CARD**

MMC Card Capacity <=2Gbytes

MMC_HIGH_CAPACITY_CARD

MMC Card Capacity>2Gbytes and <2Tbytes

Exported Constants

BLOCKSIZE

Block size is 512 bytes

MMC Exported Macros

__HAL_MMC_ENABLE_IT

Description:

- Enable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
 - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
 - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
 - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
 - SDMMC_IT_DTIMEOUT: Data timeout interrupt
 - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
 - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
 - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
 - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
 - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
 - SDMMC_IT_DHOLD: Data transfer Hold interrupt
 - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
 - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
 - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
 - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
 - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
 - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
 - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
 - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
 - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
 - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
 - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
 - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
 - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_DISABLE_IT

Description:

- Disable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:

- SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
- SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
- SDMMC_IT_CTIMEOUT: Command response timeout interrupt
- SDMMC_IT_DTIMEOUT: Data timeout interrupt
- SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
- SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
- SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
- SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
- SDMMC_IT_DHOLD: Data transfer Hold interrupt
- SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
- SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
- SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
- SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
- SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
- SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
- SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
- SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
- SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
- SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

[__HAL_MMC_GET_FLAG](#)**Description:**

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

Parameters:

- __HANDLE__: MMC Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
 - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
 - SDMMC_FLAG_CTIMEOUT: Command response timeout
 - SDMMC_FLAG_DTIMEOUT: Data timeout
 - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
 - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
 - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
 - SDMMC_FLAG_CMDSENT: Command sent (no response required)
 - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
 - SDMMC_FLAG_DHOLD: Data transfer Hold
 - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
 - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
 - SDMMC_FLAG_CPSMACT: Command path state machine active
 - SDMMC_FLAG_DPSMACT: Data path state machine active
 - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
 - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full

- SDMMC_FLAG_TXFIFO: Transmit FIFO full
- SDMMC_FLAG_RXFIFO: Receive FIFO full
- SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
- SDMMC_FLAG_RXFIFOE: Receive FIFO empty
- SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
- SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
- SDMMC_FLAG_SDIOIT: SD I/O interrupt received
- SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
- SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
- SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
- SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
- SDMMC_FLAG_IDMATE: IDMA transfer error
- SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The new state of MMC FLAG (SET or RESET).

[__HAL_MMC_CLEAR_FLAG](#)**Description:**

- Clear the MMC's pending flags.

Parameters:

- __HANDLE__: MMC Handle
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
 - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
 - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
 - SDMMC_FLAG_CTIMEOUT: Command response timeout
 - SDMMC_FLAG_DTIMEOUT: Data timeout
 - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
 - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
 - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
 - SDMMC_FLAG_CMDSENT: Command sent (no response required)
 - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
 - SDMMC_FLAG_DHOLD: Data transfer Hold
 - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
 - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
 - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
 - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
 - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
 - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
 - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
 - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
 - SDMMC_FLAG_IDMATE: IDMA transfer error
 - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

[__HAL_MMC_GET_IT](#)**Description:**

- Check whether the specified MMC interrupt has occurred or not.

Parameters:

- `__HANDLE__`: MMC Handle
- `__INTERRUPT__`: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
 - `SDMMC_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
 - `SDMMC_IT_DCRCFAIL`: Data block sent/received (CRC check failed) interrupt
 - `SDMMC_IT_CTIMEOUT`: Command response timeout interrupt
 - `SDMMC_IT_DTIMEOUT`: Data timeout interrupt
 - `SDMMC_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
 - `SDMMC_IT_RXOVERR`: Received FIFO overrun error interrupt
 - `SDMMC_IT_CMDREND`: Command response received (CRC check passed) interrupt
 - `SDMMC_IT_CMDSENT`: Command sent (no response required) interrupt
 - `SDMMC_IT_DATAEND`: Data end (data counter, DATACOUNT, is zero) interrupt
 - `SDMMC_IT_DHOLD`: Data transfer Hold interrupt
 - `SDMMC_IT_DBCKEND`: Data block sent/received (CRC check passed) interrupt
 - `SDMMC_IT_DABORT`: Data transfer aborted by CMD12 interrupt
 - `SDMMC_IT_DPSMACT`: Data path state machine active interrupt
 - `SDMMC_IT_CPSMACT`: Command path state machine active interrupt
 - `SDMMC_IT_TXFIFOHE`: Transmit FIFO Half Empty interrupt
 - `SDMMC_IT_RXFIFOHF`: Receive FIFO Half Full interrupt
 - `SDMMC_IT_TXFIFOF`: Transmit FIFO full interrupt
 - `SDMMC_IT_RXFIFOF`: Receive FIFO full interrupt
 - `SDMMC_IT_TXFIFOE`: Transmit FIFO empty interrupt
 - `SDMMC_IT_RXFIFOE`: Receive FIFO empty interrupt
 - `SDMMC_IT_BUSYD0`: Inverted value of SDMMC_D0 line (Busy)
 - `SDMMC_IT_BUSYD0END`: End of SDMMC_D0 Busy following a CMD response detected interrupt
 - `SDMMC_IT_SDIOIT`: SD I/O interrupt received interrupt
 - `SDMMC_IT_ACKFAIL`: Boot Acknowledgment received interrupt
 - `SDMMC_IT_ACKTIMEOUT`: Boot Acknowledgment timeout interrupt
 - `SDMMC_IT_VSWEND`: Voltage switch critical timing section completion interrupt
 - `SDMMC_IT_CKSTOP`: SDMMC_CK stopped in Voltage switch procedure interrupt
 - `SDMMC_IT_IDMATE`: IDMA transfer error interrupt
 - `SDMMC_IT_IDMABTC`: IDMA buffer transfer complete interrupt

Return value:

- The: new state of MMC IT (SET or RESET).

[__HAL_MMC_CLEAR_IT](#)

Description:

- Clear the MMC's interrupt pending bits.

Parameters:

- `__HANDLE__`: MMC Handle
- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
 - `SDMMC_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
 - `SDMMC_IT_DCRCFAIL`: Data block sent/received (CRC check failed) interrupt
 - `SDMMC_IT_CTIMEOUT`: Command response timeout interrupt
 - `SDMMC_IT_DTIMEOUT`: Data timeout interrupt
 - `SDMMC_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
 - `SDMMC_IT_RXOVERR`: Received FIFO overrun error interrupt

- SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
- SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
- SDMMC_IT_DHOLD: Data transfer Hold interrupt
- SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
- SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
- SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
- SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
- SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
- SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
- SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
- SDMMC_IT_IDMATE: IDMA transfer error interrupt
- SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

MMC Handle Structure definition[**MMC_InitTypeDef**](#)[**MMC_TypeDef**](#)

45 HAL MMC Extension Driver

45.1 MMCEEx Firmware driver API description

45.1.1 How to use this driver

The MMC Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using `HAL_MMCEEx_ConfigDMAMultiBuffer()` function.
- Start Read and Write for multibuffer mode using `HAL_MMCEEx_ReadBlocksDMAMultiBuffer()` and `HAL_MMCEEx_WriteBlocksDMAMultiBuffer()` functions.

45.1.2 Multibuffer functions

This section provides functions allowing to configure the multibuffer mode and start read and write multibuffer mode for MMC HAL driver.

This section contains the following APIs:

- [`HAL_MMCEEx_ConfigDMAMultiBuffer`](#)
- [`HAL_MMCEEx_ReadBlocksDMAMultiBuffer`](#)
- [`HAL_MMCEEx_WriteBlocksDMAMultiBuffer`](#)
- [`HAL_MMCEEx_ChangeDMABuffer`](#)
- [`HAL_MMCEEx_Read_DMADoubleBuffer0CpltCallback`](#)
- [`HAL_MMCEEx_Read_DMADoubleBuffer1CpltCallback`](#)
- [`HAL_MMCEEx_Write_DMADoubleBuffer0CpltCallback`](#)
- [`HAL_MMCEEx_Write_DMADoubleBuffer1CpltCallback`](#)

45.1.3 Detailed description of functions

`HAL_MMCEEx_ConfigDMAMultiBuffer`

Function name

`HAL_StatusTypeDef HAL_MMCEEx_ConfigDMAMultiBuffer (MMC_HandleTypeDef * hmmc, uint32_t * pDataBuffer0, uint32_t * pDataBuffer1, uint32_t BufferSize)`

Function description

Configure DMA Dual Buffer mode.

Parameters

- **hmmc:** MMC handle
- **pDataBuffer0:** Pointer to the buffer0 that will contain/receive the transferred data
- **pDataBuffer1:** Pointer to the buffer1 that will contain/receive the transferred data
- **BufferSize:** Size of Buffer0 in Blocks. Buffer0 and Buffer1 must have the same size.

Return values

- **HAL:** status

`HAL_MMCEEx_ReadBlocksDMAMultiBuffer`

Function name

`HAL_StatusTypeDef HAL_MMCEEx_ReadBlocksDMAMultiBuffer (MMC_HandleTypeDef * hmmc, uint32_t BlockAdd, uint32_t NumberOfBlocks)`

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** MMC handle
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Total number of blocks to read

Return values

- **HAL:** status

HAL_MMCEEx_WriteBlocksDMAMultiBuffer

Function name

HAL_StatusTypeDef HAL_MMCEEx_WriteBlocksDMAMultiBuffer (MMC_HandleTypeDef * hmmc, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Write block(s) to a specified address in a card.

Parameters

- **hmmc:** MMC handle
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Total number of blocks to read

Return values

- **HAL:** status

HAL_MMCEEx_ChangeDMABuffer

Function name

HAL_StatusTypeDef HAL_MMCEEx_ChangeDMABuffer (MMC_HandleTypeDef * hmmc, HAL_MMCEEx_DMABuffer_MemoryTypeDef Buffer, uint32_t * pDataBuffer)

Function description

Change the DMA Buffer0 or Buffer1 address on the fly.

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure.
- **Buffer:** the buffer to be changed, This parameter can be one of the following values:
MMC_DMA_BUFFER0 or MMC_DMA_BUFFER1
- **pDataBuffer:** The new address

Return values

- **HAL:** status

Notes

- The BUFFER0 address can be changed only when the current transfer use BUFFER1 and the BUFFER1 address can be changed only when the current transfer use BUFFER0.

HAL_MMCEEx_Read_DMADoubleBuffer0CpltCallback

Function name

void HAL_MMCEEx_Read_DMADoubleBuffer0CpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Read DMA Buffer 0 Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

HAL_MMCEEx_Read_DMADoubleBuffer1CpltCallback

Function name

void HAL_MMCEEx_Read_DMADoubleBuffer1CpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Read DMA Buffer 1 Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

HAL_MMCEEx_Write_DMADoubleBuffer0CpltCallback

Function name

void HAL_MMCEEx_Write_DMADoubleBuffer0CpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Write DMA Buffer 0 Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

HAL_MMCEEx_Write_DMADoubleBuffer1CpltCallback

Function name

void HAL_MMCEEx_Write_DMADoubleBuffer1CpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Write DMA Buffer 1 Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

46 HAL NAND Generic Driver

46.1 NAND Firmware driver registers structures

46.1.1 NAND_IDTypeDef

Data Fields

- *uint8_t Maker_Id*
- *uint8_t Device_Id*
- *uint8_t Third_Id*
- *uint8_t Fourth_Id*

Field Documentation

- *uint8_t NAND_IDTypeDef::Maker_Id*
- *uint8_t NAND_IDTypeDef::Device_Id*
- *uint8_t NAND_IDTypeDef::Third_Id*
- *uint8_t NAND_IDTypeDef::Fourth_Id*

46.1.2 NAND_AddressTypeDef

Data Fields

- *uint16_t Page*
- *uint16_t Plane*
- *uint16_t Block*

Field Documentation

- *uint16_t NAND_AddressTypeDef::Page*
NAND memory Page address
- *uint16_t NAND_AddressTypeDef::Plane*
NAND memory Zone address
- *uint16_t NAND_AddressTypeDef::Block*
NAND memory Block address

46.1.3 NAND_DeviceConfigTypeDef

Data Fields

- *uint32_t PageSize*
- *uint32_t SpareAreaSize*
- *uint32_t BlockSize*
- *uint32_t BlockNbr*
- *uint32_t PlaneNbr*
- *uint32_t PlaneSize*
- *FunctionalState ExtraCommandEnable*

Field Documentation

- ***uint32_t NAND_DeviceConfigTypeDef::PageSize***
NAND memory page (without spare area) size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::SpareAreaSize***
NAND memory spare area size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::BlockSize***
NAND memory block size measured in number of pages
- ***uint32_t NAND_DeviceConfigTypeDef::BlockNbr***
NAND memory number of total blocks
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneNbr***
NAND memory number of planes
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneSize***
NAND memory zone size measured in number of blocks
- ***FunctionalState NAND_DeviceConfigTypeDef::ExtraCommandEnable***
NAND extra command needed for Page reading mode. This parameter is mandatory for some NAND parts after the read command (NAND_CMD_AREA_TRUE1) and before DATA reading sequence. Example: Toshiba THTH58BYG3S0HBAI6. This parameter could be ENABLE or DISABLE Please check the Read Mode sequence in the NAND device datasheet

46.1.4 NAND_HandleTypeDef

Data Fields

- ***FMC_NAND_TypeDef * Instance***
- ***FMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***_IO HAL_NAND_StateTypeDef State***
- ***NAND_DeviceConfigTypeDef Config***

Field Documentation

- ***FMC_NAND_TypeDef* NAND_HandleTypeDef::Instance***
Register base address
- ***FMC_NAND_InitTypeDef NAND_HandleTypeDef::Init***
NAND device control configuration parameters
- ***HAL_LockTypeDef NAND_HandleTypeDef::Lock***
NAND locking object
- ***_IO HAL_NAND_StateTypeDef NAND_HandleTypeDef::State***
NAND device access state
- ***NAND_DeviceConfigTypeDef NAND_HandleTypeDef::Config***
NAND physical characteristic information structure

46.2 NAND Firmware driver API description

46.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC/FSMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.
- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page_8b()/HAL_NAND_Read_SpareArea_8b(), HAL_NAND_Write_Page_8b()/HAL_NAND_Write_SpareArea_8b(), HAL_NAND_Read_Page_16b()/HAL_NAND_Read_SpareArea_16b(), HAL_NAND_Write_Page_16b()/HAL_NAND_Write_SpareArea_16b() to read/write page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the NAND_DeviceConfigTypeDef structure. The read/write address information is contained by the Nand_Address_Typedef structure passed as parameter.
- Perform NAND flash Reset chip operation using the function HAL_NAND_Reset().
- Perform NAND flash erase block operation using the function HAL_NAND_Erase_Block(). The erase block address information is contained in the Nand_Address_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL_NAND_Read_Status().
- You can also control the NAND device by calling the control APIs HAL_NAND_ECC_Enable() / HAL_NAND_ECC_Disable() to respectively enable/disable the ECC code correction feature or the function HAL_NAND_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL_NAND_GetState()

Note: *This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.*

46.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

This section contains the following APIs:

- [HAL_NAND_Init](#)
- [HAL_NAND_DelInit](#)
- [HAL_NAND_MspInit](#)
- [HAL_NAND_MspDelInit](#)
- [HAL_NAND_IRQHandler](#)
- [HAL_NAND_ITCallback](#)

46.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

This section contains the following APIs:

- [HAL_NAND_Read_ID](#)
- [HAL_NAND_Reset](#)
- [HAL_NAND_ConfigDevice](#)
- [HAL_NAND_Read_Page_8b](#)
- [HAL_NAND_Read_Page_16b](#)
- [HAL_NAND_Write_Page_8b](#)
- [HAL_NAND_Write_Page_16b](#)
- [HAL_NAND_Read_SpareArea_8b](#)
- [HAL_NAND_Read_SpareArea_16b](#)
- [HAL_NAND_Write_SpareArea_8b](#)
- [HAL_NAND_Write_SpareArea_16b](#)
- [HAL_NAND_Erase_Block](#)
- [HAL_NAND_Address_Inc](#)

46.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

This section contains the following APIs:

- [*HAL_NAND_ECC_Enable*](#)
- [*HAL_NAND_ECC_Disable*](#)
- [*HAL_NAND_GetECC*](#)

46.2.5 NAND State functions

This subsection permits to get in run-time the status of the NAND controller and the data flow.

This section contains the following APIs:

- [*HAL_NAND_GetState*](#)
- [*HAL_NAND_Read_Status*](#)

46.2.6 Detailed description of functions

HAL_NAND_Init

Function name

```
HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hñand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)
```

Function description

Perform NAND memory Initialization sequence.

Parameters

- **hñand**: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ComSpace_Timing**: pointer to Common space timing structure
- **AttSpace_Timing**: pointer to Attribute space timing structure

Return values

- **HAL**: status

HAL_NAND_Delinit

Function name

```
HAL_StatusTypeDef HAL_NAND_Delinit (NAND_HandleTypeDef * hñand)
```

Function description

Perform NAND memory De-Initialization sequence.

Parameters

- **hñand**: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL**: status

HAL_NAND_MspInit

Function name

```
void HAL_NAND_MspInit (NAND_HandleTypeDef * hñand)
```

Function description

NAND MSP Init.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_MspInit

Function name

void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP Delinit.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_IRQHandler

Function name

void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)

Function description

This function handles NAND device interrupt request.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ITCallback

Function name

void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)

Function description

NAND interrupt feature callback.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_Read_ID

Function name

```
HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hñand, NAND_IDTypeDef * pNAND_ID)
```

Function description

Read the NAND memory electronic signature.

Parameters

- **hñand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pNAND_ID:** NAND ID structure

Return values

- **HAL:** status

HAL_NAND_ConfigDevice

Function name

```
HAL_StatusTypeDef HAL_NAND_ConfigDevice (NAND_HandleTypeDef * hñand,  
NAND_DeviceConfigTypeDef * pDeviceConfig)
```

Function description

Configure the device: Enter the physical parameters of the device.

Parameters

- **hñand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pDeviceConfig:** : pointer to NAND_DeviceConfigTypeDef structure

Return values

- **HAL:** status

HAL_NAND_Reset

Function name

```
HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hñand)
```

Function description

NAND memory reset.

Parameters

- **hñand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_Read_Page_8b

Function name

```
HAL_StatusTypeDef HAL_NAND_Read_Page_8b (NAND_HandleTypeDef * hñand,  
NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)
```

Function description

Read Page(s) from NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to destination read buffer
- **NumPageToRead:** : number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_8b

Function name

**HAL_StatusTypeDef HAL_NAND_Write_Page_8b (NAND_HandleTypeDef * hnand,
NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)**

Function description

Write Page(s) to NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to source buffer to write
- **NumPageToWrite:** : number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_8b

Function name

**HAL_StatusTypeDef HAL_NAND_Read_SpareArea_8b (NAND_HandleTypeDef * hnand,
NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)**

Function description

Read Spare area(s) from NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_8b

Function name

**HAL_StatusTypeDef HAL_NAND_Write_SpareArea_8b (NAND_HandleTypeDef * hnand,
NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)**

Function description

Write Spare area(s) to NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to source buffer to write
- **NumSpareAreaTowrite:** : number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Read_Page_16b

Function name

**HAL_StatusTypeDef HAL_NAND_Read_Page_16b (NAND_HandleTypeDef * hnand,
NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToRead)**

Function description

Read Page(s) from NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to destination read buffer. pBuffer should be 16bits aligned
- **NumPageToRead:** : number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_16b

Function name

**HAL_StatusTypeDef HAL_NAND_Write_Page_16b (NAND_HandleTypeDef * hnand,
NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToWrite)**

Function description

Write Page(s) to NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to source buffer to write. pBuffer should be 16bits aligned
- **NumPageToWrite:** : number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_16b

Function name

**HAL_StatusTypeDef HAL_NAND_Read_SpareArea_16b (NAND_HandleTypeDef * hnand,
NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaToRead)**

Function description

Read Spare area(s) from NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_16b

Function name

**HAL_StatusTypeDef HAL_NAND_Write_SpareArea_16b (NAND_HandleTypeDef * hnand,
NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaTowrite)**

Function description

Write Spare area(s) to NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure
- **pBuffer:** : pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaTowrite:** : number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Erase_Block

Function name

**HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef *
pAddress)**

Function description

NAND memory Block erase.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure

Return values

- **HAL:** status

HAL_NAND_Address_Inc

Function name

uint32_t HAL_NAND_Address_Inc (NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

Increment the NAND memory address.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **The:** new status of the increment address operation. It can be:
 - NAND_VALID_ADDRESS: When the new address is valid address
 - NAND_INVALID_ADDRESS: When the new address is invalid address

HAL_NAND_ECC_Enable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)

Function description

Enables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ECC_Disable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)

Function description

Disables dynamically FMC_NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_GetECC

Function name

HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)

Function description

Disables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ECCval:** pointer to ECC value
- **Timeout:** maximum timeout to wait

Return values

- **HAL:** status

HAL_NAND_GetState**Function name**

HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hndl)

Function description

return the NAND state

Parameters

- **hndl:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** state

HAL_NAND_Read_Status**Function name**

uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hndl)

Function description

NAND memory read status.

Parameters

- **hndl:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **NAND:** status

46.3 NAND Firmware driver defines

46.3.1 NAND

NAND Exported Macros**_HAL_NAND_RESET_HANDLE_STATE****Description:**

- Reset NAND handle state.

Parameters:

- **_HANDLE_**: specifies the NAND handle.

Return value:

- None

47 HAL NOR Generic Driver

47.1 NOR Firmware driver registers structures

47.1.1 NOR_IDTypeDef

Data Fields

- `uint16_t Manufacturer_Code`
- `uint16_t Device_Code1`
- `uint16_t Device_Code2`
- `uint16_t Device_Code3`

Field Documentation

- `uint16_t NOR_IDTypeDef::Manufacturer_Code`

Defines the device's manufacturer code used to identify the memory.

- `uint16_t NOR_IDTypeDef::Device_Code1`

- `uint16_t NOR_IDTypeDef::Device_Code2`

- `uint16_t NOR_IDTypeDef::Device_Code3`

Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set. They can also be accessed by issuing an Auto Select command.

47.1.2 NOR_CFITypeDef

Data Fields

- `uint16_t CFI_1`
- `uint16_t CFI_2`
- `uint16_t CFI_3`
- `uint16_t CFI_4`

Field Documentation

- `uint16_t NOR_CFITypeDef::CFI_1`

< Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory

- `uint16_t NOR_CFITypeDef::CFI_2`

- `uint16_t NOR_CFITypeDef::CFI_3`

- `uint16_t NOR_CFITypeDef::CFI_4`

47.1.3 NOR_HandleTypeDef

Data Fields

- `FMC_NORSRAM_TypeDef * Instance`
- `FMC_NORSRAM_EXTENDED_TypeDef * Extended`
- `FMC_NORSRAM_InitTypeDef Init`

- **`HAL_LockTypeDef Lock`**
- **`_IO HAL_NOR_StateTypeDef State`**

Field Documentation

- **`FMC_NORSRAM_TypeDef* NOR_HandleTypeDef::Instance`**
Register base address
- **`FMC_NORSRAM_EXTENDED_TypeDef* NOR_HandleTypeDef::Extended`**
Extended mode register base address
- **`FMC_NORSRAM_InitTypeDef NOR_HandleTypeDef::Init`**
NOR device control configuration parameters
- **`HAL_LockTypeDef NOR_HandleTypeDef::Lock`**
NOR locking object
- **`_IO HAL_NOR_StateTypeDef NOR_HandleTypeDef::State`**
NOR device access state

47.2 NOR Firmware driver API description

47.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function `HAL_NOR_Init()` with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function `HAL_NOR_Read_ID()`. The read information is stored in the `NOR_ID_TypeDef` structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions `HAL_NOR_Read()`, `HAL_NOR_Program()`.
- Perform NOR flash erase block/chip operations using the functions `HAL_NOR_Erase_Block()` and `HAL_NOR_Erase_Chip()`.
- Read the NOR flash CFI (common flash interface) IDs using the function `HAL_NOR_Read_CFI()`. The read information is stored in the `NOR_CFI_TypeDef` structure declared by the function caller.
- You can also control the NOR device by calling the control APIs `HAL_NOR_WriteOperation_Enable()`/ `HAL_NOR_WriteOperation_Disable()` to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function `HAL_NOR_GetState()`

Note:

This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- `NOR_WRITE` : NOR memory write data to specified address

47.2.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- `HAL_NOR_Init`
- `HAL_NOR_DelInit`
- `HAL_NOR_MspInit`

- [*HAL_NOR_MspDeInit*](#)
- [*HAL_NOR_MspWait*](#)

47.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

This section contains the following APIs:

- [*HAL_NOR_Read_ID*](#)
- [*HAL_NOR_ReturnToReadMode*](#)
- [*HAL_NOR_Read*](#)
- [*HAL_NOR_Program*](#)
- [*HAL_NOR_ReadBuffer*](#)
- [*HAL_NOR_ProgramBuffer*](#)
- [*HAL_NOR_Erase_Block*](#)
- [*HAL_NOR_Erase_Chip*](#)
- [*HAL_NOR_Read_CFI*](#)

47.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

This section contains the following APIs:

- [*HAL_NOR_WriteOperation_Enable*](#)
- [*HAL_NOR_WriteOperation_Disable*](#)

47.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

This section contains the following APIs:

- [*HAL_NOR_GetState*](#)
- [*HAL_NOR_GetStatus*](#)

47.2.6 Detailed description of functions

HAL_NOR_Init

Function name

```
HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef *  
Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)
```

Function description

Perform the NOR memory Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timing:** pointer to NOR control timing structure
- **ExtTiming:** pointer to NOR extended mode timing structure

Return values

- **HAL:** status

HAL_NOR_Delnit

Function name

`HAL_StatusTypeDef HAL_NOR_Delnit (NOR_HandleTypeDef * hnor)`

Function description

Perform NOR memory De-Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_MspInit

Function name

`void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)`

Function description

NOR MSP Init.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspDelnit

Function name

`void HAL_NOR_MspDelnit (NOR_HandleTypeDef * hnor)`

Function description

NOR MSP Delnit.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspWait

Function name

`void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)`

Function description

NOR MSP Wait for Ready/Busy signal.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

- **Timeout:** Maximum timeout value

Return values

- **None:**

HAL_NOR_Read_ID

Function name

HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)

Function description

Read NOR flash IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_ID:** : pointer to NOR ID structure

Return values

- **HAL:** status

HAL_NOR_ReturnToReadMode

Function name

HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)

Function description

Returns the NOR memory to Read mode.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_Read

Function name

HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Read data from NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** pointer to Device address
- **pData:** : pointer to read data

Return values

- **HAL:** status

HAL_NOR_Program

Function name

```
HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)
```

Function description

Program data to NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** Device address
- **pData:** : pointer to the data to write

Return values

- **HAL:** status

HAL_NOR_ReadBuffer

Function name

```
HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)
```

Function description

Reads a half-word buffer from the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal address to read from.
- **pData:** pointer to the buffer that receives the data read from the NOR memory.
- **uwBufferSize:** : number of Half word to read.

Return values

- **HAL:** status

HAL_NOR_ProgramBuffer

Function name

```
HAL_StatusTypeDef HAL_NOR_ProgramBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)
```

Function description

Writes a half-word buffer to the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal start write address
- **pData:** pointer to source data buffer.
- **uwBufferSize:** Size of the buffer to write

Return values

- **HAL:** status

HAL_NOR_Erase_Block

Function name

```
HAL_StatusTypeDef HAL_NOR_Erase_Block (NOR_HandleTypeDef * hnor, uint32_t BlockAddress,  
uint32_t Address)
```

Function description

Erase the specified block of the NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **BlockAddress:** : Block to erase address
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Erase_Chip

Function name

```
HAL_StatusTypeDef HAL_NOR_Erase_Chip (NOR_HandleTypeDef * hnor, uint32_t Address)
```

Function description

Erase the entire NOR chip.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** : Device address

Return values

- **HAL:** status

HAL_NOR_Read_CFI

Function name

```
HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)
```

Function description

Read NOR flash CFI IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_CFI:** : pointer to NOR CFI IDs structure

Return values

- **HAL:** status

HAL_NOR_WriteOperation_Enable

Function name

```
HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)
```

Function description

Enables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)

Function description

Disables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_GetState

Function name

HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)

Function description

return the NOR controller state

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **NOR:** controller state

HAL_NOR_GetStatus

Function name

HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)

Function description

Returns the NOR operation status.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address
- **Timeout:** NOR programming Timeout

Return values

- **NOR_Status:** The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT

47.3 NOR Firmware driver defines

47.3.1 NOR

NOR Exported Macros

[__HAL_NOR_RESET_HANDLE_STATE](#)

Description:

- Reset NOR handle state.

Parameters:

- [__HANDLE__](#): specifies the NOR handle.

Return value:

- None

48 HAL OPAMP Generic Driver

48.1 OPAMP Firmware driver registers structures

48.1.1 OPAMP_InitTypeDef

Data Fields

- *uint32_t PowerMode*
- *uint32_t Mode*
- *uint32_t InvertingInput*
- *uint32_t NonInvertingInput*
- *uint32_t PgaGain*
- *uint32_t PgaConnect*
- *uint32_t UserTrimming*
- *uint32_t TrimmingValueP*
- *uint32_t TrimmingValueN*
- *uint32_t TrimmingValuePHighSpeed*
- *uint32_t TrimmingValueNHighSpeed*

Field Documentation

- *uint32_t OPAMP_InitTypeDef::PowerMode*

Specifies the power mode Normal or High Speed. This parameter must be a value of **OPAMP PowerMode**

- *uint32_t OPAMP_InitTypeDef::Mode*

Specifies the OPAMP mode This parameter must be a value of **OPAMP Mode** mode is either Standalone, - Follower or PGA

- *uint32_t OPAMP_InitTypeDef::InvertingInput*

Specifies the inverting input in Standalone & PGA modes

- In Standalone mode i.e when mode is OPAMP_STANDALONE_MODE This parameter must be a value of **OPAMP Inverting Input**
- In Follower mode i.e when mode is OPAMP_FOLLOWER_MODE & In PGA mode i.e when mode is OPAMP_PGA_MODE This parameter is Not Applicable

- *uint32_t OPAMP_InitTypeDef::NonInvertingInput*

Specifies the non inverting input of the opamp: This parameter must be a value of **OPAMP Non Inverting Input**

- *uint32_t OPAMP_InitTypeDef::PgaGain*

Specifies the gain in PGA mode i.e. when mode is OPAMP_PGA_MODE. This parameter must be a value of **OPAMP Pga Gain**

- *uint32_t OPAMP_InitTypeDef::PgaConnect*

Specifies the inverting pin in PGA mode i.e. when mode is OPAMP_PGA_MODE This parameter must be a value of **OPAMP Pga Connect** Either: not connected, connected to VINM0, connected to VINM1 (VINM0 or VINM1 are typically used for external filtering)

- *uint32_t OPAMP_InitTypeDef::UserTrimming*

Specifies the trimming mode This parameter must be a value of **OPAMP User Trimming** UserTrimming is either factory or user trimming.

- *uint32_t OPAMP_InitTypeDef::TrimmingValueP*

Specifies the offset trimming value (PMOS) in Normal Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value

- ***uint32_t OPAMP_InitTypeDef::TrimmingValueN***

Specifies the offset trimming value (NMOS) in Normal Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value

- ***uint32_t OPAMP_InitTypeDef::TrimmingValuePHighSpeed***

Specifies the offset trimming value (PMOS) in High Speed Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value

- ***uint32_t OPAMP_InitTypeDef::TrimmingValueNHighSpeed***

Specifies the offset trimming value (NMOS) in High Speed Mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31. 16 is typical default value

48.1.2 OPAMP_HandleTypeDef

Data Fields

- ***OPAMP_TypeDef * Instance***
- ***OPAMP_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_OPAMP_StateTypeDef State***

Field Documentation

- ***OPAMP_TypeDef* OPAMP_HandleTypeDef::Instance***
OPAMP instance's registers base address
- ***OPAMP_InitTypeDef OPAMP_HandleTypeDef::Init***
OPAMP required parameters
- ***HAL_StatusTypeDef OPAMP_HandleTypeDef::Status***
OPAMP peripheral status
- ***HAL_LockTypeDef OPAMP_HandleTypeDef::Lock***
Locking object
- ***__IO HAL_OPAMP_StateTypeDef OPAMP_HandleTypeDef::State***
OPAMP communication state

48.2 OPAMP Firmware driver API description

48.2.1 OPAMP Peripheral Features

The device integrates 2 operational amplifiers OPAMP1 & OPAMP2

1. The OPAMP(s) provides several exclusive running modes.
 - Standalone mode
 - Programmable Gain Amplifier (PGA) modes
 - Follower mode
2. Each OPAMP(s) can be configured in normal and high speed mode.
3. The OPAMP(s) provide(s) calibration capabilities.

- Calibration aims at correcting some offset for running mode.
 - The OPAMP uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
 - The user defined settings can be figured out using self calibration handled by HAL_OPAMP_SelfCalibrate, HAL_OPAMPEx_SelfCalibrateAll
 - HAL_OPAMP_SelfCalibrate:
 - Runs automatically the calibration in 2 steps. (90% of VDDA for NMOS transistors, 10% of VDDA for PMOS transistors). (As OPAMP is Rail-to-rail input/output, these 2 steps calibration is appropriate and enough in most cases).
 - Runs automatically the calibration.
 - Enables the user trimming mode
 - Updates the init structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)
 - HAL_OPAMPEx_SelfCalibrateAll runs calibration of all OPAMPs in parallel to save search time.
4. Running mode: Standalone mode
 - Gain is set externally (gain depends on external loads).
 - Follower mode also possible externally by connecting the inverting input to the output.
 5. Running mode: Follower mode
 - No Inverting Input is connected.
 6. Running mode: Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
 7. The OPAMP(s) output(s) can be internally connected to resistor feedback output.
 8. OPAMP gain can be selected as :
 - a. Gain of x2, x4, x8 or x16 for non inverting mode with:
 - VREF- referenced.
 - Filtering on VINM0, VREF- referenced.
 - VINM0 node for bias voltage and VINP0 for input signal.
 - VINM0 node for bias voltage and VINP0 for input signal, VINM1 node for filtering.
 - b. Gain of x-1, x-3, x-7 or x-15 for inverting mode with:
 - VINM0 node for input signal and VINP0 for bias.
 - VINM0 node for input signal and VINP0 for bias voltage, VINM1 node for filtering.
 9. The OPAMPs inverting input can be selected according to the Reference Manual "OPAMP functional description" chapter.
 10. The OPAMPs non inverting input can be selected according to the Reference Manual "OPAMP functional description" chapter.

48.2.2 How to use this driver

High speed / normal power mode

To run in high speed mode:

1. Configure the OPAMP using HAL_OPAMP_Init() function:
 - Select OPAMP_POWERMODE_HIGHSPEED
 - Otherwise select OPAMP_POWERMODE_NORMAL

Calibration

To run the OPAMP calibration self calibration:

1. Start calibration using HAL_OPAMP_SelfCalibrate. Store the calibration results.

Running mode

To use the OPAMP, perform the following steps:

1. Fill in the HAL_OPAMP_MspInit() to
 - Enable the OPAMP Peripheral clock using macro __HAL_RCC_OPAMP_CLK_ENABLE()
 - Configure the OPAMP input AND output in analog mode using HAL_GPIO_Init() to map the OPAMP output to the GPIO pin.
2. Configure the OPAMP using HAL_OPAMP_Init() function:
 - Select the mode
 - Select the inverting input
 - Select the non-inverting input
 - If PGA mode is enabled, Select if inverting input is connected.
 - Select either factory or user defined trimming mode.
 - If the user-defined trimming mode is enabled, select PMOS & NMOS trimming values (typically values set by HAL_OPAMP_SelfCalibrate function).
3. Enable the OPAMP using HAL_OPAMP_Start() function.
4. Disable the OPAMP using HAL_OPAMP_Stop() function.
5. Lock the OPAMP in running mode using HAL_OPAMP_Lock() function. Caution: On STM32H7, HAL OPAMP lock is software lock only (not hardware lock as on some other STM32 devices)
6. If needed, unlock the OPAMP using HAL_OPAMPEx_Unlock() function.

Running mode: change of configuration while OPAMP ON

To Re-configure OPAMP when OPAMP is ON (change on the fly)

1. If needed, fill in the HAL_OPAMP_MspInit()
 - This is the case for instance if you wish to use new OPAMP I/O
2. Configure the OPAMP using HAL_OPAMP_Init() function:
 - As in configure case, select first the parameters you wish to modify.
3. Change from high speed mode to normal power mode (& vice versa) requires first HAL_OPAMP_DelInit() (force OPAMP OFF) and then HAL_OPAMP_Init(). In other words, of OPAMP is ON, HAL_OPAMP_Init can NOT change power mode alone.

48.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- [HAL_OPAMP_Init](#)
- [HAL_OPAMP_DelInit](#)
- [HAL_OPAMP_MspInit](#)
- [HAL_OPAMP_MspDelInit](#)

48.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the OPAMP start, stop and calibration actions.

This section contains the following APIs:

- [HAL_OPAMP_Start](#)
- [HAL_OPAMP_Stop](#)
- [HAL_OPAMP_SelfCalibrate](#)

48.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the OPAMP data transfers.

This section contains the following APIs:

- [HAL_OPAMP_Lock](#)
- [HAL_OPAMP_GetTrimOffset](#)

48.2.6 Peripheral State functions

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

This section contains the following APIs:

- [HAL_OPAMP_GetState](#)

48.2.7 Detailed description of functions

HAL_OPAMP_Init

Function name

HAL_StatusTypeDef HAL_OPAMP_Init (OPAMP_HandleTypeDef * hopamp)

Function description

Initialize the OPAMP according to the specified parameters in the OPAMP_InitTypeDef and initialize the associated handle.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- If the selected opamp is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_OPAMP_DeInit

Function name

HAL_StatusTypeDef HAL_OPAMP_DeInit (OPAMP_HandleTypeDef * hopamp)

Function description

Deinitialize the OPAMP peripheral.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- Deinitialization can be performed if the OPAMP configuration is locked. (the lock is SW in H7)

HAL_OPAMP_MspInit

Function name

void HAL_OPAMP_MspInit (OPAMP_HandleTypeDef * hopamp)

Function description

Initialize the OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_MspDelInit**Function name****void HAL_OPAMP_MspDelInit (OPAMP_HandleTypeDef * hopamp)****Function description**

DeInitialize OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_Start**Function name****HAL_StatusTypeDef HAL_OPAMP_Start (OPAMP_HandleTypeDef * hopamp)****Function description**

Start the OPAMP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_Stop**Function name****HAL_StatusTypeDef HAL_OPAMP_Stop (OPAMP_HandleTypeDef * hopamp)****Function description**

Stop the OPAMP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_SelfCalibrate**Function name****HAL_StatusTypeDef HAL_OPAMP_SelfCalibrate (OPAMP_HandleTypeDef * hopamp)****Function description**

Run the self calibration of one OPAMP.

Parameters

- **hopamp:** handle

Return values

- **Updated:** offset trimming values (PMOS & NMOS), user trimming is enabled
- **HAL:** status

Notes

- Calibration is performed in the mode specified in OPAMP init structure (mode normal or high-speed). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated.

HAL_OPAMP_Lock

Function name

HAL_StatusTypeDef HAL_OPAMP_Lock (OPAMP_HandleTypeDef * hopamp)

Function description

Lock the selected OPAMP configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- On STM32H7, HAL OPAMP lock is software lock only (in contrast of hardware lock available on some other STM32 devices)

HAL_OPAMP_GetTrimOffset

Function name

HAL_OPAMP_TrimmingValueTypeDef HAL_OPAMP_GetTrimOffset (OPAMP_HandleTypeDef * hopamp, uint32_t trimmingoffset)

Function description

Return the OPAMP factory trimming value.

Parameters

- **hopamp:** : OPAMP handle
- **trimmingoffset:** : Trimming offset (P or N) This parameter must be a value of OPAMP Factory Trimming

Return values

- **Trimming:** value (P or N): range: 0->31 or OPAMP_FACTORYTRIMMING_DUMMY if trimming value is not available

Notes

- On STM32H7 OPAMP, user can retrieve factory trimming if OPAMP has never been set to user trimming before. Therefore, this function must be called when OPAMP init parameter "UserTrimming" is set to trimming factory, and before OPAMP calibration (function "HAL_OPAMP_SelfCalibrate()"). Otherwise, factory trimming value cannot be retrieved and error status is returned.
- Calibration parameter retrieved is corresponding to the mode specified in OPAMP init structure (mode normal or high-speed). To retrieve calibration parameters for both modes, repeat this function after OPAMP init structure accordingly updated.

HAL_OPAMP_GetState

Function name

HAL_OPAMP_StateTypeDef HAL_OPAMP_GetState (OPAMP_HandleTypeDef * hopamp)

Function description

Return the OPAMP handle state.

Parameters

- **hopamp:** : OPAMP handle

Return values

- **HAL:** state

48.3 OPAMP Firmware driver defines

48.3.1 OPAMP

OPAMP Exported Macros

__HAL_OPAMP_RESET_HANDLE_STATE

Description:

- Reset OPAMP handle state.

Parameters:

- **__HANDLE__**: OPAMP handle.

Return value:

- None

OPAMP Factory Trimming

OPAMP_FACTORYTRIMMING_DUMMY

Dummy value if trimming value could not be retrieved

OPAMP_FACTORYTRIMMING_N

Offset trimming N

OPAMP_FACTORYTRIMMING_P

Offset trimming P

OPAMP Inverting Input

OPAMP_INVERTINGINPUT_IO0

OPAMP inverting input connected to dedicated IO pin

OPAMP_INVERTINGINPUT_IO1

OPAMP inverting input connected to dedicated IO pin

OPAMP Mode

OPAMP_STANDALONE_MODE

standalone mode

OPAMP_PGA_MODE

PGA mode

OPAMP_FOLLOWER_MODE

follower mode

OPAMP Non Inverting Input

OPAMP_NONINVERTINGINPUT_IO0

OPAMP non-inverting input connected to dedicated IO pin

OPAMP_NONINVERTINGINPUT_DAC_CH

OPAMP non-inverting input connected internally to DAC channel

OPAMP Pga Connect**OPAMP_PGA_CONNECT_INVERTINGINPUT_NO**

In PGA mode, the inverting input is not connected

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0

In PGA mode, the inverting input is connected to VINM0

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0_BIAS

In PGA mode, the inverting input is connected to VINM0 or bias

OPAMP_PGA_CONNECT_INVERTINGINPUT_IO0_IO1_BIAS

In PGA mode, the inverting input is connected to VINM0 or bias , VINM1 connected for filtering

OPAMP Pga Gain**OPAMP_PGA_GAIN_2_OR_MINUS_1**

PGA gain could be 2 or -1

OPAMP_PGA_GAIN_4_OR_MINUS_3

PGA gain could be 4 or -3

OPAMP_PGA_GAIN_8_OR_MINUS_7

PGA gain could be 8 or -7

OPAMP_PGA_GAIN_16_OR_MINUS_15

PGA gain could be 16 or -15

OPAMP PowerMode**OPAMP_POWERMODE_NORMAL****OPAMP_POWERMODE_HIGHSPEED*****OPAMP User Trimming*****OPAMP_TRIMMING_FACTORY**

Factory trimming

OPAMP_TRIMMING_USER

User trimming

OPAMP VREF**OPAMP_VREF_3VDDA**

OPAMP Vref = 3.3% VDDA

OPAMP_VREF_10VDDA

OPAMP Vref = 10% VDDA

OPAMP_VREF_50VDDA

OPAMP Vref = 50% VDDA

OPAMP_VREF_90VDDA

OPAMP Vref = 90% VDDA

49 HAL OPAMP Extension Driver

49.1 OPAMPEx Firmware driver API description

49.1.1 Extended IO operation functions

- OPAMP Self calibration.

49.1.2 Peripheral Control functions

- OPAMP unlock.

This section contains the following APIs:

- *HAL_OPAMPEx_Unlock*

49.1.3 Detailed description of functions

`HAL_OPAMPEx_SelfCalibrateAll`

Function name

`HAL_StatusTypeDef HAL_OPAMPEx_SelfCalibrateAll (OPAMP_HandleTypeDef * hopamp1,
OPAMP_HandleTypeDef * hopamp2)`

Function description

Run the self calibration of 2 OPAMPs in parallel.

Parameters

- `hopamp1`: handle
- `hopamp2`: handle

Return values

- `HAL`: status

Notes

- Trimming values (PMOS & NMOS) are updated and user trimming is enabled if calibration is successful.
- Calibration is performed in the mode specified in OPAMP init structure (mode normal or low power). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated.

`HAL_OPAMPEx_Unlock`

Function name

`HAL_StatusTypeDef HAL_OPAMPEx_Unlock (OPAMP_HandleTypeDef * hopamp)`

Function description

Unlock the selected OPAMP configuration.

Parameters

- `hopamp`: OPAMP handle

Return values

- `HAL`: status

Notes

- This function must be called only when OPAMP is in state "locked".

50 HAL PCD Generic Driver

50.1 PCD Firmware driver registers structures

50.1.1 PCD_HandleTypeDef

Data Fields

- *PCD_TypeDef * Instance*
- *PCD_InitTypeDef Init*
- *PCD_EPTTypeDef IN_ep*
- *PCD_EPTTypeDef OUT_ep*
- *HAL_LockTypeDef Lock*
- *_IO PCD_StateTypeDef State*
- *uint32_t Setup*
- *PCD_LPM_StateTypeDef LPM_State*
- *uint32_t BESL*
- *uint32_t lpm_active*
- *uint32_t battery_charging_active*
- *void * pData*

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance***
Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init***
PCD required parameters
- ***PCD_EPTTypeDef PCD_HandleTypeDef::IN_ep[16]***
IN endpoint parameters
- ***PCD_EPTTypeDef PCD_HandleTypeDef::OUT_ep[16]***
OUT endpoint parameters
- ***HAL_LockTypeDef PCD_HandleTypeDef::Lock***
PCD peripheral status
- ***_IO PCD_StateTypeDef PCD_HandleTypeDef::State***
PCD communication state
- ***uint32_t PCD_HandleTypeDef::Setup[12]***
Setup packet buffer
- ***PCD_LPM_StateTypeDef PCD_HandleTypeDef::LPM_State***
LPM State
- ***uint32_t PCD_HandleTypeDef::BESL***
- ***uint32_t PCD_HandleTypeDef::lpm_active***
Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_HandleTypeDef::battery_charging_active***
Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE

- ***void* PCD_HandleTypeDef::pData***

Pointer to upper stack Handler

50.2 PCD Firmware driver API description

50.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in PCD handle
3. Call HAL_PCD_Init() API to initialize the PCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
 - a. Enable the PCD/USB Low Level interface clock using
 - __OTGFS-OTG_CLK_ENABLE() / __OTGHS-OTG_CLK_ENABLE();
 - __OTGHSULPI_CLK_ENABLE(); (For High Speed Mode)
 - b. Initialize the related GPIO clocks
 - c. Configure PCD pin-out
 - d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
 - a. hpcd.pData = pdev;
6. Enable PCD transmission and reception:
 - a. HAL_PCD_Start();

50.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- [**HAL_PCD_Init**](#)
- [**HAL_PCD_DelInit**](#)
- [**HAL_PCD_MspInit**](#)
- [**HAL_PCD_MspDelInit**](#)

50.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.

This section contains the following APIs:

- [**HAL_PCD_Start**](#)
- [**HAL_PCD_Stop**](#)
- [**HAL_PCD_IRQHandler**](#)
- [**HAL_PCD_DataOutStageCallback**](#)
- [**HAL_PCD_DataInStageCallback**](#)
- [**HAL_PCD_SetupStageCallback**](#)
- [**HAL_PCD_SOFCallback**](#)
- [**HAL_PCD_ResetCallback**](#)
- [**HAL_PCD_SuspendCallback**](#)
- [**HAL_PCD_ResumeCallback**](#)
- [**HAL_PCD_ISOOUTIncompleteCallback**](#)
- [**HAL_PCD_ISOINIncompleteCallback**](#)
- [**HAL_PCD_ConnectCallback**](#)

- [*HAL_PCD_DisconnectCallback*](#)

50.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

This section contains the following APIs:

- [*HAL_PCD_DevConnect*](#)
- [*HAL_PCD_DevDisconnect*](#)
- [*HAL_PCD_SetAddress*](#)
- [*HAL_PCD_EP_Open*](#)
- [*HAL_PCD_EP_Close*](#)
- [*HAL_PCD_EP_Receive*](#)
- [*HAL_PCD_EP_GetRxCount*](#)
- [*HAL_PCD_EP_Transmit*](#)
- [*HAL_PCD_EP_SetStall*](#)
- [*HAL_PCD_EP_ClrStall*](#)
- [*HAL_PCD_EP_Flush*](#)
- [*HAL_PCD_ActivateRemoteWakeup*](#)
- [*HAL_PCD_DeActivateRemoteWakeup*](#)

50.2.5 Peripheral State functions

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

This section contains the following APIs:

- [*HAL_PCD_GetState*](#)

50.2.6 Detailed description of functions

HAL_PCD_Init

Function name

HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and create the associated handle.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeInit

Function name

HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes the PCD peripheral.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_MspInit**Function name****void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)****Function description**

Initializes the PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_MspDeInit**Function name****void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)****Function description**

DeInitializes PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_Start**Function name****HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)****Function description**

Start The USB OTG Device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Stop**Function name****HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)****Function description**

Stop The USB OTG Device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_IRQHandler**Function name**

```
void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)
```

Function description

Handle PCD interrupt request.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DataOutStageCallback**Function name**

```
void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epcnum)
```

Function description

Data out stage callback.

Parameters

- **hpcd:** PCD handle
- **epcnum:** endpoint number

Return values

- **None:**

HAL_PCD_DataInStageCallback**Function name**

```
void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epcnum)
```

Function description

Data IN stage callback.

Parameters

- **hpcd:** PCD handle
- **epcnum:** endpoint number

Return values

- **None:**

HAL_PCD_SetupStageCallback**Function name**

```
void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)
```

Function description

Setup stage callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SOFCallback

Function name

void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Start Of Frame callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResetCallback

Function name

void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Reset callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SuspendCallback

Function name

void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)

Function description

Suspend event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResumeCallback

Function name

void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)

Function description

Resume event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ISOOUTIncompleteCallback

Function name

void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epcnum)

Function description

Incomplete ISO OUT callback.

Parameters

- **hpcd:** PCD handle
- **epcnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOINIncompleteCallback

Function name

void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epcnum)

Function description

Incomplete ISO IN callback.

Parameters

- **hpcd:** PCD handle
- **epcnum:** endpoint number

Return values

- **None:**

HAL_PCD_ConnectCallback

Function name

void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Connection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DisconnectCallback

Function name

void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Disconnection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DevConnect**Function name****HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)****Function description**

Connect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DevDisconnect**Function name****HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)****Function description**

Disconnect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SetAddress**Function name****HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)****Function description**

Set the USB Device address.

Parameters

- **hpcd:** PCD handle
- **address:** new device address

Return values

- **HAL:** status

HAL_PCD_EP_Open**Function name****HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)****Function description**

Open and configure an endpoint.

Parameters

- **hpcd:** PCD handle

- **ep_addr:** endpoint address
- **ep_mps:** endpoint max packet size
- **ep_type:** endpoint type

Return values

- **HAL:** status

HAL_PCD_EP_Close

Function name

```
HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)
```

Function description

Deactivate an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Receive

Function name

```
HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf,  
uint32_t len)
```

Function description

Receive an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the reception buffer
- **len:** amount of data to be received

Return values

- **HAL:** status

HAL_PCD_EP_Transmit

Function name

```
HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t *  
pBuf, uint32_t len)
```

Function description

Send an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the transmission buffer
- **len:** amount of data to be sent

Return values

- **HAL:** status

HAL_PCD_EP_GetRxCount

Function name

`uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)`

Function description

Get Received Data Size.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **Data:** Size

HAL_PCD_EP_SetStall

Function name

`HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)`

Function description

Set a STALL condition over an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_ClrStall

Function name

`HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)`

Function description

Clear a STALL condition over in an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Flush

Function name

`HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)`

Function description

Flush an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_ActivateRemoteWakeups

Function name

HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeups (PCD_HandleTypeDef * hpcd)

Function description

Activate remote wake-up signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeActivateRemoteWakeups

Function name

HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeups (PCD_HandleTypeDef * hpcd)

Function description

De-activate remote wake-up signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_GetState

Function name

PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)

Function description

Return the PCD handle state.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** state

50.3 PCD Firmware driver defines

50.3.1 PCD

PCD Exported Macros

_HAL_PCD_ENABLE

`_HAL_PCD_DISABLE`
`_HAL_PCD_GET_FLAG`
`_HAL_PCD_CLEAR_FLAG`
`_HAL_PCD_IS_INVALID_INTERRUPT`

`_HAL_PCD_UNGATE_PHYCLOCK`
`_HAL_PCD_GATE_PHYCLOCK`
`_HAL_PCD_IS_PHY_SUSPENDED`

`USB_OTG_FS_WAKEUP_EXTI_RISING_EDGE`
`USB_OTG_FS_WAKEUP_EXTI_FALLING_EDGE`
`USB_OTG_FS_WAKEUP_EXTI_RISING_FALLING_EDGE`
`USB_OTG_HS_WAKEUP_EXTI_RISING_EDGE`
`USB_OTG_HS_WAKEUP_EXTI_FALLING_EDGE`
`USB_OTG_HS_WAKEUP_EXTI_RISING_FALLING_EDGE`
`USB_OTG_HS_WAKEUP_EXTI_LINE`

External interrupt line 43 Connected to the USB HS EXTI Line

`USB_OTG_FS_WAKEUP_EXTI_LINE`

External interrupt line 44 Connected to the USB FS EXTI Line

`_HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_IT`
`_HAL_USB_OTG_HS_WAKEUP_EXTI_DISABLE_IT`
`_HAL_USB_OTG_HS_WAKEUP_EXTI_GET_FLAG`
`_HAL_USB_OTG_HS_WAKEUP_EXTI_CLEAR_FLAG`
`_HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_RISING_EDGE`
`_HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_FALLING_EDGE`
`_HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE`
`_HAL_USB_OTG_HS_WAKEUP_EXTI_GENERATE_SWIT`
`_HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_IT`
`_HAL_USB_OTG_FS_WAKEUP_EXTI_DISABLE_IT`
`_HAL_USB_OTG_FS_WAKEUP_EXTI_GET_FLAG`
`_HAL_USB_OTG_FS_WAKEUP_EXTI_CLEAR_FLAG`

`_HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_RISING_EDGE`
`_HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_FALLING_EDGE`
`_HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE`
`_HAL_USB_OTG_FS_WAKEUP_EXTI_GENERATE_SWIT`

PCD Instance definition

`IS_PCD_ALL_INSTANCE`

PCD PHY Module

`PCD_PHY_ULPI`

`PCD_PHY_EMBEDDED`

PCD Speed

`PCD_SPEED_HIGH`

`PCD_SPEED_HIGH_IN_FULL`

`PCD_SPEED_FULL`

Turnaround Timeout Value

`USBD_HS_TRDT_VALUE`

`USBD_FS_TRDT_VALUE`

51 HAL PCD Extension Driver

51.1 PCDEEx Firmware driver API description

51.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- [*HAL_PCDEx_SetTxFiFo*](#)
- [*HAL_PCDEx_SetRxFiFo*](#)
- [*HAL_PCDEx_ActivateLPM*](#)
- [*HAL_PCDEx_DeActivateLPM*](#)
- [*HAL_PCDEx_BCD_VBUSDetect*](#)
- [*HAL_PCDEx_ActivateBCD*](#)
- [*HAL_PCDEx_DeActivateBCD*](#)
- [*HAL_PCDEx_BCD_Callback*](#)
- [*HAL_PCDEx_LPM_Callback*](#)

51.1.2 Detailed description of functions

`HAL_PCDEx_SetTxFiFo`

Function name

`HAL_StatusTypeDef HAL_PCDEx_SetTxFiFo (PCD_HandleTypeDef * hpcd, uint8_t fifo, uint16_t size)`

Function description

Set Tx FIFO.

Parameters

- **hpcd:** PCD handle
- **fifo:** The number of Tx fifo
- **size:** Fifo size

Return values

- **HAL:** status

`HAL_PCDEx_SetRxFiFo`

Function name

`HAL_StatusTypeDef HAL_PCDEx_SetRxFiFo (PCD_HandleTypeDef * hpcd, uint16_t size)`

Function description

Set Rx FIFO.

Parameters

- **hpcd:** PCD handle
- **size:** Size of Rx fifo

Return values

- **HAL:** status

HAL_PCDEx_ActivateLPM**Function name****HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)****Function description**

Activate LPM Feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateLPM**Function name****HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM (PCD_HandleTypeDef * hpcd)****Function description**

De-activate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_ActivateBCD**Function name****HAL_StatusTypeDef HAL_PCDEx_ActivateBCD (PCD_HandleTypeDef * hpcd)****Function description**

Activate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateBCD**Function name****HAL_StatusTypeDef HAL_PCDEx_DeActivateBCD (PCD_HandleTypeDef * hpcd)****Function description**

Deactivate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_BCD_VBUSDetect**Function name**

void HAL_PCDEx_BCD_VBUSDetect (PCD_HandleTypeDef * hpcd)

Function description

Handle Battery Charging Process.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_LPM_Callback**Function name**

void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)

Function description

Send LPM message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

HAL_PCDEx_BCD_Callback**Function name**

void HAL_PCDEx_BCD_Callback (PCD_HandleTypeDef * hpcd, PCD_BCD_MsgTypeDef msg)

Function description

Send BatteryCharging message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

52 HAL PWR Generic Driver

52.1 PWR Firmware driver registers structures

52.1.1 PWR_PVDTTypeDef

Data Fields

- *uint32_t PVDLevel*
- *uint32_t Mode*

Field Documentation

- *uint32_t PWR_PVDTTypeDef::PVDLevel*

PVDLevel: Specifies the PVD detection level. This parameter can be a value of **PWR PVD detection level**

- *uint32_t PWR_PVDTTypeDef::Mode*

Mode: Specifies the operating mode for the selected pins. This parameter can be a value of **PWR PVD Mode**

52.2 PWR Firmware driver API description

52.2.1 Initialization and De-Initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses. To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__HAL_RCC_PWR_CLK_ENABLE()` macro.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.

This section contains the following APIs:

- [**HAL_PWR_DelInit**](#)
- [**HAL_PWR_EnableBkUpAccess**](#)
- [**HAL_PWR_DisableBkUpAccess**](#)

52.2.2 Peripheral Control functions

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[7:0] bits in the PWR_CR1 register).
- A PVDO flag is available to indicate if VDD is higher or lower than the PVD threshold. This event is internally connected to the EXTI line 16 to generate an interrupt if enabled. It is configurable through `__HAL_PWR_PVD_EXTI_ENABLE_IT()` macro.
- The PVD is stopped in Standby mode.

Wake-up pin configuration

- Wake-up pin is used to wake up the system from Standby mode. The pin pull is configurable through the WKUPEPR register to be in No pull-up, Pull-up and Pull-down. The pin polarity is configurable through the WKUPEPR register to be active on rising or falling edges.
- There are up to six Wake-up pin in the STM32H7 devices family.

Low Power modes configuration

The device present 3 principles low-power modes features:

- SLEEP mode: Cortex-M7 core stopped and D1, D2 and D3 peripherals kept running.
- STOP mode: all clocks are stopped and the regulator running in main or low power mode.
- STANDBY mode: D1, D2 and D3 domains enter DSTANDBY mode and the VCORE supply regulator is powered off.

SLEEP mode

- Entry: The Sleep mode is entered by using the HAL_PWR_EnterSLEEPMode(Regulator, SLEEPEntry) function.
 - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
 - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction

Note:

The Regulator parameter is not used for the STM32H7 family and is kept as parameter just to maintain compatibility with the lower power families (STM32L).

- Exit: Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

STOP mode

In system Stop mode, all clocks in the 1.2V domain are stopped, the PLL, the HSI, and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. The voltage regulator can be configured either in normal or low-power mode. To minimize the consumption In Stop mode, FLASH can be powered off before entering the Stop mode using the HAL_PWREx_EnableFlashPowerDown() function. It can be switched on again by software after exiting the Stop mode using the HAL_PWREx_DisableFlashPowerDown() function.

- Entry: The Stop mode is entered using the HAL_PWR_EnterSTOPMode(Regulator, STOPEntry) function with:
 - Regulator:
 - PWR_MAINREGULATOR_ON: Main regulator ON.
 - PWR_LOWPOWERREGULATOR_ON: Low Power regulator ON.
 - STOPEntry:
 - PWR_STOPENTRY_WFI: enter STOP mode with WFI instruction
 - PWR_STOPENTRY_WFE: enter STOP mode with WFE instruction
- Exit: Any EXTI Line (Internal or External) configured in Interrupt/Event mode.

STANDBY mode

- The system Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M deep sleep mode, with the voltage regulator disabled. The system is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers, backup SRAM and Standby circuitry.

The voltage regulator is OFF. (++) Entry: (+++) The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function. (++) Exit: (+++) WKUP pin rising or falling edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wakeup mode).
- RTC auto-wakeup (AWU) from the STOP and STANDBY modes
 - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL_RTC_SetAlarm_IT() function.

- To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL_RTCEx_SetTimeStamp_IT() or HAL_RTCEx_SetTamper_IT() functions.
- To wake up from the Stop mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL_RTCEx_SetWakeUpTimer_IT() function.

This section contains the following APIs:

- [HAL_PWR_ConfigPVD](#)
- [HAL_PWR_EnablePVD](#)
- [HAL_PWR_DisablePVD](#)
- [HAL_PWR_EnableWakeUpPin](#)
- [HAL_PWR_DisableWakeUpPin](#)
- [HAL_PWR_EnterSLEEPMode](#)
- [HAL_PWR_EnterSTOPMode](#)
- [HAL_PWR_EnterSTANDBYMode](#)
- [HAL_PWR_EnableSleepOnExit](#)
- [HAL_PWR_DisableSleepOnExit](#)
- [HAL_PWR_EnableSEVOnPend](#)
- [HAL_PWR_DisableSEVOnPend](#)
- [HAL_PWR_PVD_IRQHandler](#)
- [HAL_PWR_PVDCallback](#)

52.2.3 Detailed description of functions

HAL_PWR_DeInit

Function name

```
void HAL_PWR_DeInit (void )
```

Function description

Deinitialize the HAL PWR peripheral registers to their default reset values.

Return values

- **None:**

Notes

- This functionality is not available in this product. The prototype is kept just to maintain compatibility with other products.

HAL_PWR_EnableBkUpAccess

Function name

```
void HAL_PWR_EnableBkUpAccess (void )
```

Function description

Enable access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).

Return values

- **None:**

Notes

- If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_DisableBkUpAccess

Function name

```
void HAL_PWR_DisableBkUpAccess (void )
```

Function description

Disable access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).

Return values

- **None:**

Notes

- If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.

HAL_PWR_ConfigPVD

Function name

```
void HAL_PWR_ConfigPVD (PWR_PVDTTypeDef * sConfigPVD)
```

Function description

Configure the voltage threshold detected by the Power Voltage Detector(PVD).

Parameters

- **sConfigPVD:** pointer to an PWR_PVDTTypeDef structure that contains the configuration information for the PVD.

Return values

- **None:**

Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

HAL_PWR_EnablePVD

Function name

```
void HAL_PWR_EnablePVD (void )
```

Function description

Enable the Power Voltage Detector(PVD).

Return values

- **None:**

HAL_PWR_DisablePVD

Function name

```
void HAL_PWR_DisablePVD (void )
```

Function description

Disable the Power Voltage Detector(PVD).

Return values

- **None:**

HAL_PWR_EnableWakeUpPin

Function name

```
void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinPolarity)
```

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPinPolarity:** Specifies which Wake-Up pin to enable. This parameter can be one of the following legacy values, which sets the default: polarity detection on high level (rising edge):
 - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5, PWR_WAKEUP_PIN6 or one of the following values where the user can explicitly states the enabled pin and the chosen polarity.
 - PWR_WAKEUP_PIN1_HIGH or PWR_WAKEUP_PIN1_LOW
 - PWR_WAKEUP_PIN2_HIGH or PWR_WAKEUP_PIN2_LOW
 - PWR_WAKEUP_PIN3_HIGH or PWR_WAKEUP_PIN3_LOW
 - PWR_WAKEUP_PIN4_HIGH or PWR_WAKEUP_PIN4_LOW
 - PWR_WAKEUP_PIN5_HIGH or PWR_WAKEUP_PIN5_LOW
 - PWR_WAKEUP_PIN6_HIGH or PWR_WAKEUP_PIN6_LOW

Return values

- **None:**

Notes

- PWR_WAKEUP_PINx and PWR_WAKEUP_PINx_HIGH are equivalent.

HAL_PWR_DisableWakeUpPin

Function name

```
void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)
```

Function description

Disable the WakeUp PINx functionality.

Parameters

- **WakeUpPinx:** Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:
 - PWR_WAKEUP_PIN1
 - PWR_WAKEUP_PIN2
 - PWR_WAKEUP_PIN3
 - PWR_WAKEUP_PIN4
 - PWR_WAKEUP_PIN5
 - PWR_WAKEUP_PIN6

Return values

- **None:**

HAL_PWR_EnterSTOPMode

Function name

```
void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)
```

Function description

Enter the system to STOP mode.

Parameters

- **Regulator:** Specifies the regulator state in Stop mode. This parameter can be one of the following values:
 - PWR_MAINREGULATOR_ON: Stop mode with regulator ON
 - PWR_LOWPOWERREGULATOR_ON: Stop mode with low power regulator ON
- **STOPEntry:** Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
 - PWR_STOPENTRY_WFI: Enter Stop mode with WFI instruction
 - PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction

Return values

- **None:**

Notes

- In System Stop mode, all I/O pins keep the same state as in Run mode.
- When exiting System Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as default system wakeup clock.
- In System STOP mode, when the voltage regulator operates in low power mode, an additional startup delay is incurred when the system is waking up. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

HAL_PWR_EnterSLEEPMode

Function name

```
void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)
```

Function description

Enter CM7 core to Sleep mode.

Parameters

- **Regulator:** Specifies the regulator state in SLEEP mode. This parameter can be one of the following values:
 - PWR_MAINREGULATOR_ON: SLEEP mode with regulator ON
 - PWR_LOWPOWERREGULATOR_ON: SLEEP mode with low power regulator ON
- **SLEEPEntry:** Specifies if SLEEP mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
 - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
 - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction

Return values

- **None:**

Notes

- This parameter is not used for the STM32H7 family and is kept as parameter just to maintain compatibility with the lower power families.

HAL_PWR_EnterSTANDBYMode

Function name

```
void HAL_PWR_EnterSTANDBYMode (void )
```

Function description

Enter the system to STANDBY mode.

Return values

- **None.:**

Notes

- The system enters Standby mode only when the D1, D2 and D3 domains are in DStandby.
- When the System exit STANDBY mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.

HAL_PWR_PVD_IRQHandler**Function name**

```
void HAL_PWR_PVD_IRQHandler (void )
```

Function description

This function handles the PWR PVD interrupt request.

Return values

- **None:**

Notes

- This API should be called under the PVD_IRQHandler().

HAL_PWR_PVDCallback**Function name**

```
void HAL_PWR_PVDCallback (void )
```

Function description

PWR PVD interrupt callback.

Return values

- **None:**

HAL_PWR_EnableSleepOnExit**Function name**

```
void HAL_PWR_EnableSleepOnExit (void )
```

Function description

Indicate Sleep-On-Exit when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit**Function name**

```
void HAL_PWR_DisableSleepOnExit (void )
```

Function description

Disable Sleep-On-Exit feature when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name

```
void HAL_PWR_EnableSEVOnPend (void )
```

Function description

Enable CORTEX SEVONPEND bit.

Return values

- **None:**

Notes

- Sets SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

HAL_PWR_DisableSEVOnPend

Function name

```
void HAL_PWR_DisableSEVOnPend (void )
```

Function description

Disable CORTEX SEVONPEND bit.

Return values

- **None:**

Notes

- Clears SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

52.3 PWR Firmware driver defines

52.3.1 PWR

PWR Enable WUP Mask

PWR_EWUP_MASK

PWR Exported Macro

_HAL_PWR_VOLTAGESCALING_CONFIG

Description:

- macros configure the main internal regulator output voltage.

Parameters:

- **_REGULATOR_**: specifies the regulator output voltage to achieve a tradeoff between performance and power consumption when the device does not operate at the maximum frequency (refer to the datasheets for more details). This parameter can be one of the following values:
 - PWR_REGULATOR_VOLTAGE_SCALE1: Regulator voltage output Scale 1 mode

- PWR_REGULATOR_VOLTAGE_SCALE2: Regulator voltage output Scale 2 mode
- PWR_REGULATOR_VOLTAGE_SCALE3: Regulator voltage output Scale 3 mode

Return value:

- None

_HAL_PWR_GET_FLAG

Description:

- Check PWR PVD/AVD and VOSflags are set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - PWR_FLAG_PVDO: PVD Output. This flag is valid only if PVD is enabled by the HAL_PWR_EnablePVD() function. The PVD is stopped by Standby mode For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.
 - PWR_FLAG_AVDO: AVD Output. This flag is valid only if AVD is enabled by the HAL_PWREx_EnableAVD() function. The AVD is stopped by Standby mode For this reason, this bit is equal to 0 after Standby or reset until the AVDE bit is set.
 - PWR_FLAG_ACTVOSRDY: This flag indicates that the Regulator voltage scaling output selection is ready.
 - PWR_FLAG_VOSRDY: This flag indicates that the Regulator voltage scaling output selection is ready.
 - PWR_FLAG_BRR: Backup regulator ready flag. This bit is not reset when the device wakes up from Standby mode or by a system reset or power reset.
 - PWR_FLAG_SB: StandBy flag
 - PWR_FLAG_STOP: STOP flag
 - PWR_FLAG_SB_D1: StandBy D1 flag
 - PWR_FLAG_SB_D2: StandBy D2 flag

Return value:

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

_HAL_PWR_CLEAR_FLAG

Description:

- Clear the PWR's flags.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - PWR_FLAG_SB: StandBy flag.
 - PWR_CPU_FLAGS: Clear STOPF, SBF, SBF_D1, and SBF_D2 CPU flags.

Return value:

- None.

_HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable the PVD EXTI Line 16.

Return value:

- None.

_HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable the PVD EXTI Line 16.

Return value:

- None.

[__HAL_PWR_PVD_EXTI_ENABLE_EVENT](#)**Description:**

- Enable event on PVD EXTI Line 16.

Return value:

- None.

[__HAL_PWR_PVD_EXTI_DISABLE_EVENT](#)**Description:**

- Disable event on PVD EXTI Line 16.

Return value:

- None.

[__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE](#)**Description:**

- Enable the PVD Extended Interrupt Rising Trigger.

Return value:

- None.

[__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE](#)**Description:**

- Disable the PVD Extended Interrupt Rising Trigger.

Return value:

- None.

[__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE](#)**Description:**

- Enable the PVD Extended Interrupt Falling Trigger.

Return value:

- None.

[__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE](#)**Description:**

- Disable the PVD Extended Interrupt Falling Trigger.

Return value:

- None.

[__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE](#)**Description:**

- PVD EXTI line configuration: set rising & falling edge trigger.

Return value:

- None.

[__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE](#)**Description:**

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

_HAL_PWR_PVD_EXTI_GET_FLAG**Description:**

- Check whether the specified PVD EXTI interrupt flag is set or not.

Return value:

- EXTI: PVD Line Status.

_HAL_PWR_PVD_EXTI_CLEAR_FLAG**Description:**

- Clear the PVD EXTI flag.

Return value:

- None.

_HAL_PWR_PVD_EXTI_GENERATE_SWIT**Description:**

- Generates a Software interrupt on PVD EXTI line.

Return value:

- None.

PWR Flag**PWR_FLAG_STOP****PWR_FLAG_SB_D1****PWR_FLAG_SB_D2****PWR_FLAG_SB****PWR_FLAG_PVDO****PWR_FLAG_AVDO****PWR_FLAG_ACTVOSRDY****PWR_FLAG_ACTVOS****PWR_FLAG_BRR****PWR_FLAG_VOSRDY****PWR_FLAG_SCUEN*****PWR Private macros to check input parameters*****IS_PWR_PVD_LEVEL****IS_PWR_PVD_MODE****IS_PWR_REGULATOR**

IS_PWR_SLEEP_ENTRY

IS_PWR_STOP_ENTRY

IS_PWR_REGULATOR_VOLTAGE

PWR PVD detection level

PWR_PVDEVEL_0

PWR_PVDEVEL_1

PWR_PVDEVEL_2

PWR_PVDEVEL_3

PWR_PVDEVEL_4

PWR_PVDEVEL_5

PWR_PVDEVEL_6

PWR_PVDEVEL_7

PWR PVD EXTI Line

PWR_EXTI_LINE_PVD

< External interrupt line 16 Connected to the PVD EXTI Line

PWR PVD Mode

PWR_PVD_MODE_NORMAL

Basic mode is used

PWR_PVD_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_PVD_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_PVD_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_PVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR PVD Mode Mask

PVD_MODE_IT

PVD_MODE_EVT

PVD_RISING_EDGE

PVD_FALLING_EDGE

PVD_RISING_FALLING_EDGE

PWR Regulator state in SLEEP/STOP mode

PWR_MAINREGULATOR_ON

PWR_LOWPOWERREGULATOR_ON

PWR Regulator Voltage Scale

PWR_REGULATOR_VOLTAGE_SCALE1

PWR_REGULATOR_VOLTAGE_SCALE2

PWR_REGULATOR_VOLTAGE_SCALE3

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

PWR_SLEEPENTRY_WFE

PWR STOP mode entry

PWR_STOPENTRY_WFI

PWR_STOPENTRY_WFE

53 HAL PWR Extension Driver

53.1 PWREx Firmware driver registers structures

53.1.1 PWREx_AVDTypeDef

Data Fields

- `uint32_t AVDLevel`
- `uint32_t Mode`

Field Documentation

- `uint32_t PWREx_AVDTypeDef::AVDLevel`

AVDLevel: Specifies the AVD detection level. This parameter can be a value of [**PWREx AVD detection level**](#)

- `uint32_t PWREx_AVDTypeDef::Mode`

Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [**PWREx AVD Mode**](#)

53.1.2 PWREx_WakeupPinTypeDef

Data Fields

- `uint32_t WakeUpPin`
- `uint32_t PinPolarity`
- `uint32_t PinPull`

Field Documentation

- `uint32_t PWREx_WakeupPinTypeDef::WakeUpPin`

WakeUpPin: Specifies the Wake-Up pin to be enabled. This parameter can be a value of [**PWREx Wake-Up Pins**](#)

- `uint32_t PWREx_WakeupPinTypeDef::PinPolarity`

PinPolarity: Specifies the Wake-Up pin polarity. This parameter can be a value of [**PWREx Pin Polarity configuration**](#)

- `uint32_t PWREx_WakeupPinTypeDef::PinPull`

PinPull: Specifies the Wake-Up pin pull. This parameter can be a value of [**PWREx Pin Pull configuration**](#)

53.2 PWREx Firmware driver API description

53.2.1 Power supply control functions

Power supply configuration

When the system is powered on, the POR monitors VDD supply. Once VDD is above the POR threshold level, the voltage regulator is enabled in the default supply configuration:

- The Voltage converter output level is set at 1.0 V in accordance with the VOS3 level configured in PWR D3 domain control register (PWR_D3CR).
- The system is kept in reset mode as long as VCORE is not ok.

- Once VCORE is ok, the system is taken out of reset and the HSI oscillator is enabled.
- Once the oscillator is stable, the system is initialized: Flash memory and option bytes are loaded and the CPU starts in Run* mode.
- The software shall then initialize the system including supply configuration programming using the HAL_PWREx_ConfigSupply(SupplySource) with:
 - SupplySource:
 - PWR_LDO_SUPPLY: VCORE Power Domains are supplied from the LDO according to VOS. LDO power mode (Main, LP, Off) will follow system low-power modes.
 - PWR_EXTERNAL_SOURCE_SUPPLY: VCORE supplied from external source and LDO bypassed, voltage monitoring still active.
- Once the supply configuration has been configured, the HAL_PWREx_ConfigSupply function checks the ACTVOSRDY bit in PWR control status register 1 (PWR_CSR1) to guarantee a valid voltage levels:
 - As long as ACTVOSRDY indicates that voltage levels are invalid, the system is in limited Run* mode, write accesses to the RAMs are not permitted and VOS shall not be changed.
 - Once ACTVOSRDY indicates that voltage levels are valid, the system is in normal Run mode, write accesses to RAMs are allowed and VOS can be changed.

This section contains the following APIs:

- [HAL_PWREx_ConfigSupply](#)
- [HAL_PWREx_GetSupplyConfig](#)
- [HAL_PWREx_ControlVoltageScaling](#)
- [HAL_PWREx_GetVoltageRange](#)
- [HAL_PWREx_ControlStopModeVoltageScaling](#)
- [HAL_PWREx_GetStopModeVoltageRange](#)

53.2.2 Low power control functions

Domains Low Power modes configuration

The system present 3 principles domains (D1, D2 and D3) that can be operated in low-power modes (DSTOP or DSTANDBY mode):

- DSTOP mode to enters a domain to STOP mode:
 - D1 domain and/or D2 domain enters DSTOP mode only when the CPU subsystem is in CSTOP mode and has allocated peripheral in the domain. In DSTOP mode the domain bus matrix clock is stopped.
 - The system enters STOP mode using one of the following scenarios:
 - D1 domain enters DSTANDBY mode (powered off) and D2, D3 domains enter DSTOP mode.
 - D2 domain enters DSTANDBY mode (powered off) and D1, D3 domains enter DSTOP mode.
 - D3 domain enters DSTANDBY mode (powered off) and D1, D2 domains enter DSTOP mode.
 - D1 and D2 domains enter DSTANDBY mode (powered off) and D3 domain enters DSTOP mode.
 - D1 and D3 domains enter DSTANDBY mode (powered off) and D2 domain enters DSTOP mode.
 - D2 and D3 domains enter DSTANDBY mode (powered off) and D1 domain enters DSTOP mode.
 - D1, D2 and D3 domains enter DSTOP mode.
 - When the system enters STOP mode, the clocks are stopped and the regulator is running in main or low power mode.
 - D3 domain can be kept in Run mode regardless of the CPU status when enter STOP mode by using HAL_PWREx_ConfigD3Domain(D3State) function.
- DSTANDBY mode to enters a domain to STANDBY mode:
 - The DSTANDBY mode is entered when the PDDS_Dn bit in PWR CPU control register (PWR_CPUCR) for the Dn domain selects Standby mode.
 - The system enters STANDBY mode only when D1, D2 and D3 domains enter DSTANDBY mode. Consequently the VCORE supply regulator is powered off.

DSTOP mode

In DStop mode the domain bus matrix clock is stopped. The Flash memory can enter low-power Stop mode when it is enabled through FLPS in PWR_CR1 register. This allows a trade-off between domain DStop restart time and low power consumption.

In DStop mode domain peripherals using the LSI or LSE clock and peripherals having a kernel clock request are still able to operate.

Before entering DSTOP mode it is recommended to call SCB_CleanDCache function in order to clean the D-Cache and guarantee the data integrity for the SRAM memories.

- Entry: The DSTOP mode is entered using the HAL_PWREx_EnterSTOPMode(Regulator, STOPEntry, Domain) function with:
 - Regulator:
 - PWR_MAINREGULATOR_ON: Main regulator ON.
 - PWR_LOWPOWERREGULATOR_ON: Low Power regulator ON.
 - STOPEntry:
 - PWR_STOPENTRY_WFI: enter STOP mode with WFI instruction
 - PWR_STOPENTRY_WFE: enter STOP mode with WFE instruction
 - Domain:
 - PWR_D1_DOMAIN: Enters D1 domain to DSTOP mode.
 - PWR_D2_DOMAIN: Enters D2 domain to DSTOP mode.
 - PWR_D3_DOMAIN: Enters D3 domain to DSTOP mode.
- Exit: Any EXTI Line (Internal or External) configured in Interrupt/Event mode.

DSTANDBY mode

In DStandby mode:

- The domain bus matrix clock is stopped.
- The domain is powered down and the domain RAM and register contents are lost.

Before entering DSTANDBY mode it is recommended to call SCB_CleanDCache function in order to clean the D-Cache and guarantee the data integrity for the SRAM memories.

- Entry: The DSTANDBY mode is entered using the HAL_PWREx_EnterSTANDBYMode(Domain) function with:
 - Domain:
 - PWR_D1_DOMAIN: Enters D1 domain to DSTANDBY mode.
 - PWR_D2_DOMAIN: Enters D2 domain to DSTANDBY mode.
 - PWR_D3_DOMAIN: Enters D3 domain to DSTANDBY mode.
- Exit: WKUP pin rising or falling edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.

Keep D3 in RUN mode

D3 domain can be kept in Run mode regardless of the CPU status when enter STOP mode by using HAL_PWREx_ConfigD3Domain(D3State) function with:

- D3State:
 - PWR_D3_DOMAIN_STOP: D3 domain will follow the CPU sub-system mode.
 - PWR_D3_DOMAIN_RUN: D3 domain remains in Run mode regardless of CPU subsystem mode.

FLASH Power Down configuration

By setting the FLPS bit in the PWR_CR1 register using the HAL_PWREx_EnableFlashPowerDown() function, the Flash memory also enters power down mode when the device enters Stop mode. When the Flash memory is in power down mode, an additional startup delay is incurred when waking up from Stop mode.

Wakeup Pins configuration

Wakeup pins allow the system to exit from Standby mode. The configuration of wakeup pins is done with the HAL_PWREx_EnableWakeUpPin(sPinParams) function with:

- sPinParams: structure to enable and configure a wakeup pin:
 - WakeUpPin: Wakeup pin to be enabled.
 - PinPolarity: Wakeup pin polarity (rising or falling edge).
 - PinPull: Wakeup pin pull (no pull, pull-up or pull-down).

The wakeup pins are internally connected to the EXTI lines [55-60] to generate an interrupt if enabled. The EXTI lines configuration is done in the HAL_PWREx_EnableWakeUpPin() function.

When a wakeup pin event is received the HAL_PWREx_WAKEUP_PIN_IRQHandler is called and the appropriate flag is set in the PWR_WKUPFR register. Then in the HAL_PWREx_WAKEUP_PIN_IRQHandler function the wakeup pin flag will be cleared and the appropriate user callback will be called. The user can add his own code by customization of function pointer HAL_PWREx_WKUPx_Callback.

This section contains the following APIs:

- [HAL_PWREx_EnterSTOPMode](#)
- [HAL_PWREx_EnterSTANDBYMode](#)
- [HAL_PWREx_ConfigD3Domain](#)
- [HAL_PWREx_EnableFlashPowerDown](#)
- [HAL_PWREx_DisableFlashPowerDown](#)
- [HAL_PWREx_EnableWakeUpPin](#)
- [HAL_PWREx_DisableWakeUpPin](#)
- [HAL_PWREx_GetWakeupFlag](#)
- [HAL_PWREx_ClearWakeupFlag](#)
- [HAL_PWREx_WAKEUP_PIN_IRQHandler](#)
- [HAL_PWREx_WKUP1_Callback](#)
- [HAL_PWREx_WKUP2_Callback](#)
- [HAL_PWREx_WKUP3_Callback](#)
- [HAL_PWREx_WKUP4_Callback](#)
- [HAL_PWREx_WKUP5_Callback](#)
- [HAL_PWREx_WKUP6_Callback](#)

53.2.3

Peripherals control functions

Main and Backup Regulators configuration

- The backup domain includes 4 Kbytes of backup SRAM accessible only from the CPU, and address in 32-bit, 16-bit or 8-bit mode. Its content is retained even in Standby or VBAT mode when the low power backup regulator is enabled. It can be considered as an internal EEPROM when VBAT is always present. You can use the HAL_PWREx_EnableBkUpReg() function to enable the low power backup regulator.
- When the backup domain is supplied by VDD (analog switch connected to VDD) the backup SRAM is powered from VDD which replaces the VBAT power supply to save battery life.
- The backup SRAM is not mass erased by a tamper event. It is read protected to prevent confidential data, such as cryptographic private key, from being accessed. The backup SRAM can be erased only through the Flash interface when a protection level change from level 1 to level 0 is requested.

Note:

Refer to the description of Read protection (RDP) in the Flash programming manual.

- The main internal regulator can be configured to have a tradeoff between performance and power consumption when the device does not operate at the maximum frequency. This is done through HAL_PWREx_ControlVoltageScaling(VOS) function which configure the VOS bit in PWR_D3CR register.
- The main internal regulator can be configured to operate in Low Power mode when the system enter STOP mode to further reduce power consumption. This is done through HAL_PWREx_ControlStopModeVoltageScaling(SVOS) function which configure the SVOS bit in PWR_CR1 register. The selected SVOS4 and SVOS5 levels add an additional startup delay when exiting from system Stop mode.

Note: Refer to the product datasheets for more details.

USB Regulator configuration

- The USB transceivers are supplied from a dedicated VDD33USB supply that can be provided either by the integrated USB regulator, or by an external USB supply.
- The USB regulator is enabled by HAL_PWREx_EnableUSBReg() function, the VDD33USB is then provided from the USB regulator.
- When the USB regulator is enabled, the VDD33USB supply level detector shall be enabled through HAL_PWREx_EnableUSBVoltageDetector() function.
- The USB regulator is disabled through HAL_PWREx_DisableUSBReg() function and VDD33USB can be provided from an external supply. In this case VDD33USB and VDD50USB shall be connected together

VBAT battery charging

- When VDD is present, the external battery connected to VBAT can be charged through an internal resistance. VBAT charging can be performed either through a 5 KOhm resistor or through a 1.5 KOhm resistor.
- VBAT charging is enabled by HAL_PWREx_EnableBatteryCharging(ResistorValue) function with:
 - ResistorValue:
 - PWR_BATTERY_CHARGING_RESISTOR_5: 5 KOhm resistor.
 - PWR_BATTERY_CHARGING_RESISTOR_1_5: 1.5 KOhm resistor.
- VBAT charging is disabled by HAL_PWREx_DisableBatteryCharging() function.

This section contains the following APIs:

- [HAL_PWREx_EnableBkUpReg](#)
- [HAL_PWREx_DisableBkUpReg](#)
- [HAL_PWREx_EnableUSBReg](#)
- [HAL_PWREx_DisableUSBReg](#)
- [HAL_PWREx_EnableUSBVoltageDetector](#)
- [HAL_PWREx_DisableUSBVoltageDetector](#)
- [HAL_PWREx_EnableBatteryCharging](#)
- [HAL_PWREx_DisableBatteryCharging](#)

53.2.4

Power Monitoring functions

VBAT and Temperature supervision

- The VBAT battery voltage supply can be monitored by comparing it with two threshold levels: VBATHigh and VBATLow. VBATH flag and VBATL flags in the PWR control register 2 (PWR_CR2), indicate if VBAT is higher or lower than the threshold.
- The temperature can be monitored by comparing it with two threshold levels, TEMPHigh and TEMPLow. TEMPH and TEMPL flags, in the PWR control register 2 (PWR_CR2), indicate whether the device temperature is higher or lower than the threshold.
- The VBAT and the temperature monitoring is enabled by HAL_PWREx_EnableMonitoring() function and disabled by HAL_PWREx_DisableMonitoring() function.

- The HAL_PWREx_GetVBATLevel() function return the VBAT level which can be: PWR_VBAT_BELOW_LOW_THRESHOLD or PWR_VBAT_ABOVE_HIGH_THRESHOLD or PWR_VBAT_BETWEEN_HIGH_LOW_THRESHOLD.
- The HAL_PWREx_GetTemperatureLevel() function return the Temperature level which can be: PWR_TEMP_BELOW_LOW_THRESHOLD or PWR_TEMP_ABOVE_HIGH_THRESHOLD or PWR_TEMP_BETWEEN_HIGH_LOW_THRESHOLD.

AVD configuration

- The AVD is used to monitor the VDDA power supply by comparing it to a threshold selected by the AVD Level (ALS[3:0] bits in the PWR_CR1 register).
- A AVDO flag is available to indicate if VDDA is higher or lower than the AVD threshold. This event is internally connected to the EXTI line 16 to generate an interrupt if enabled. It is configurable through __HAL_PWR_AVD_EXTI_ENABLE_IT() macro.
- The AVD is stopped in System Standby mode.

This section contains the following APIs:

- [HAL_PWREx_EnableMonitoring](#)
- [HAL_PWREx_DisableMonitoring](#)
- [HAL_PWREx_GetTemperatureLevel](#)
- [HAL_PWREx_GetVBATLevel](#)
- [HAL_PWREx_ConfigAVD](#)
- [HAL_PWREx_EnableAVD](#)
- [HAL_PWREx_DisableAVD](#)
- [HAL_PWREx_PVD_AVD_IRQHandler](#)
- [HAL_PWREx_AVDCallback](#)

53.2.5 Detailed description of functions

HAL_PWREx_ConfigSupply

Function name

`HAL_StatusTypeDef HAL_PWREx_ConfigSupply (uint32_t SupplySource)`

Function description

Configure the system Power Supply.

Parameters

- **SupplySource:** Specifies the Power Supply source. This parameter can be one of the following values:
 - PWR_LDO_SUPPLY: The LDO supplies the Vcore Power Domains.
 - PWR_EXTERNAL_SOURCE_SUPPLY: The Vcore Power Domains are supplied from external power source. The LDO is Bypassed.

Return values

- **HAL:** status.

HAL_PWREx_GetSupplyConfig

Function name

`uint32_t HAL_PWREx_GetSupplyConfig (void)`

Function description

Get the power supply configuration.

Return values

- **The:** supply configuration.

HAL_PWREx_ControlVoltageScaling

Function name

`HAL_StatusTypeDef HAL_PWREx_ControlVoltageScaling (uint32_t VoltageScaling)`

Function description

Configure the main internal regulator output voltage.

Parameters

- **VoltageScaling:** Specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values:
 - PWR_REGULATOR_VOLTAGE_SCALE1: Regulator voltage output range 1 mode.
 - PWR_REGULATOR_VOLTAGE_SCALE2: Regulator voltage output range 2 mode.
 - PWR_REGULATOR_VOLTAGE_SCALE3: Regulator voltage output range 3 mode.

Return values

- **HAL:** Status

Notes

- When moving from Range 1 to Range 2, the system frequency must be decreased before calling `HAL_PWREx_ControlVoltageScaling()` API. When moving from Range 2 to Range 1, the system frequency can be increased after calling `HAL_PWREx_ControlVoltageScaling()` API.
- When moving from a Range to an other one, the API waits for VOSRDY flag to be set before returning the status. If the flag is not set within 1000 microseconds, `HAL_TIMEOUT` status is reported.

HAL_PWREx_GetVoltageRange

Function name

`uint32_t HAL_PWREx_GetVoltageRange (void)`

Function description

Get the main internal regulator output voltage.

Return values

- **The:** actual applied VOS for VDD11 Voltage Scaling selection.

HAL_PWREx_ControlStopModeVoltageScaling

Function name

`HAL_StatusTypeDef HAL_PWREx_ControlStopModeVoltageScaling (uint32_t VoltageScaling)`

Function description

Configure the main internal regulator output voltage in STOP mode.

Parameters

- **VoltageScaling:** Specifies the regulator output voltage when the system enters STOP mode to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values:
 - PWR_REGULATOR_SVOS_SCALE3: Regulator voltage output range 3 mode.
 - PWR_REGULATOR_SVOS_SCALE4: Regulator voltage output range 4 mode.
 - PWR_REGULATOR_SVOS_SCALE5: Regulator voltage output range 5 mode.

Return values

- **HAL:** Status

Notes

- The Stop mode voltage scaling for SVOS4 and SVOS5 sets the voltage regulator in Low-power (LP) mode to further reduce power consumption. When preselecting SVOS3, the use of the voltage regulator low-power mode (LP) can be selected by LPDS register bit.
- The selected SVOS4 and SVOS5 levels add an additional startup delay when exiting from system Stop mode.

HAL_PWREx_GetStopModeVoltageRange

Function name

```
uint32_t HAL_PWREx_GetStopModeVoltageRange (void )
```

Function description

Get the main internal regulator output voltage in STOP mode.

Return values

- **The:** actual applied VOS for VDD11 Voltage Scaling selection.

HAL_PWREx_EnterSTOPMode

Function name

```
void HAL_PWREx_EnterSTOPMode (uint32_t Regulator, uint8_t STOPENtry, uint32_t Domain)
```

Function description

Enter a Domain to DSTOP mode.

Parameters

- **Regulator:** Specifies the regulator state in Stop mode. This parameter can be one of the following values:
 - PWR_MAINREGULATOR_ON: Stop mode with regulator ON
 - PWR_LOWPOWERREGULATOR_ON: Stop mode with low power regulator ON
- **STOPENtry:** Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
 - PWR_STOPENTRY_WFI: Enter DStop mode with WFI instruction
 - PWR_STOPENTRY_WFE: Enter DStop mode with WFE instruction
- **Domain:** Specifies the Domain to enter STOP mode. This parameter can be one of the following values:
 - PWR_D1_DOMAIN: Enter D1 Domain to DSTOP mode.
 - PWR_D2_DOMAIN: Enter D2 Domain to DSTOP mode.
 - PWR_D3_DOMAIN: Enter D3 Domain to DSTOP mode.

Return values

- **None:**

Notes

- In DStop mode the domain bus matrix clock is stopped.
- The system D3 domain enters Stop mode only when the CPU subsystem is in CStop mode, the EXTI wakeup sources are inactive and at least one PDDS_Dn bit in PWR CPU control register (PWR_CPUCR) for any domain request Stop.
- In system D3 domain Stop mode, D1 domain and D2 domain are either in DStop and/or DStandby mode.
- Before entering DSTOP mode it is recommended to call SCB_CleanDCache function in order to clean the D-Cache and guarantee the data integrity for the SRAM memories.
- In System Stop mode, the domain peripherals that use the LSI or LSE clock, and the peripherals that have a kernel clock request to select HSI or CSI as source, are still able to operate.

HAL_PWREx_EnterSTANDBYMode

Function name

```
void HAL_PWREx_EnterSTANDBYMode (uint32_t Domain)
```

Function description

Enter a Domain to DSTANDBY mode.

Parameters

- **Domain:** Specifies the Domain to enter to STANDBY mode. This parameter can be one of the following values:
 - PWR_D1_DOMAIN: Enter D1 Domain to DSTANDBY mode.
 - PWR_D2_DOMAIN: Enter D2 Domain to DSTANDBY mode.
 - PWR_D3_DOMAIN: Enter D3 Domain to DSTANDBY mode.

Return values

- **None:**

Notes

- The DStandby mode is entered when all PDDS_Dn bits in PWR_CPUCR for the Dn domain select Standby mode. When the system enters Standby mode, the voltage regulator is disabled.
- When D2 or D3 domain is in DStandby mode and the CPU sets the domain PDDS_Dn bit to select Stop mode, the domain remains in DStandby mode. The domain will only exit DStandby when the CPU allocates a peripheral in the domain.
- The system D3 domain enters Standby mode only when the D1 and D2 domain are in DStandby.
- Before entering DSTANDBY mode it is recommended to call SCB_CleanDCache function in order to clean the D-Cache and guarantee the data integrity for the SRAM memories.

HAL_PWREx_ConfigD3Domain

Function name

```
void HAL_PWREx_ConfigD3Domain (uint32_t D3State)
```

Function description

Configure the D3 Domain state when the CPU is in low power mode.

Parameters

- **D3State:** Specifies the D3 state. This parameter can be one of the following values:
 - PWR_D3_DOMAIN_STOP: D3 domain will follow the CPU sub-system mode.
 - PWR_D3_DOMAIN_RUN : D3 domain will stay in RUN mode regardless of the CPU sub-system mode.

Return values

- **None:**

HAL_PWREx_EnableFlashPowerDown

Function name

```
void HAL_PWREx_EnableFlashPowerDown (void )
```

Function description

Enable the Flash Power Down in Stop mode.

Return values

- **None:**

HAL_PWREx_DisableFlashPowerDown**Function name**

```
void HAL_PWREx_DisableFlashPowerDown (void )
```

Function description

Disable the Flash Power Down in Stop mode.

Return values

- **None:**

HAL_PWREx_EnableWakeUpPin**Function name**

```
void HAL_PWREx_EnableWakeUpPin (PWREx_WakeupPinTypeDef * sPinParams)
```

Function description

Enable the Wake-up PINx functionality.

Parameters

- **sPinParams:** pointer to an PWREx_WakeupPinTypeDef structure that contains the configuration informations for the wake-up Pin.

Return values

- **None:**

HAL_PWREx_DisableWakeUpPin**Function name**

```
void HAL_PWREx_DisableWakeUpPin (uint32_t WakeUpPin)
```

Function description

Disable the Wake-up PINx functionality.

Parameters

- **WakeUpPin:** Specifies the Wake-Up pin to be disabled. This parameter can be one of the following values:
 - PWR_WAKEUP_PIN1: Disable PA0 wake-up PIN..
 - PWR_WAKEUP_PIN2: Disable PA2 wake-up PIN..
 - PWR_WAKEUP_PIN3: Disable PI8 wake-up PIN..
 - PWR_WAKEUP_PIN4: Disable PC13 wake-up PIN..
 - PWR_WAKEUP_PIN5: Disable PI11 wake-up PIN..
 - PWR_WAKEUP_PIN6: Disable PC1 wake-up PIN..

Return values

- **None:**

HAL_PWREx_GetWakeupFlag**Function name**

```
uint32_t HAL_PWREx_GetWakeupFlag (uint32_t WakeUpFlag)
```

Function description

Get the Wake-Up Pin flag.

Parameters

- **WakeUpFlag:** Specifies the Wake-Up PIN flag to check. This parameter can be one of the following values:
 - PWR_WAKEUP_FLAG1: A wakeup event was received from PA0.
 - PWR_WAKEUP_FLAG2: A wakeup event was received from PA2.
 - PWR_WAKEUP_FLAG3: A wakeup event was received from PC1.
 - PWR_WAKEUP_FLAG4: A wakeup event was received from PC13.
 - PWR_WAKEUP_FLAG5: A wakeup event was received from PI8.
 - PWR_WAKEUP_FLAG6: A wakeup event was received from PI11.

Return values

- **The:** Wake-Up pin flag.

`HAL_PWREx_ClearWakeupsFlag`

Function name

`HAL_StatusTypeDef HAL_PWREx_ClearWakeupsFlag (uint32_t WakeUpFlag)`

Function description

Clear the Wake-Up pin flag.

Parameters

- **WakeUpFlag:** Specifies the Wake-Up PIN flag to clear. This parameter can be one of the following values:
 - PWR_WAKEUP_FLAG1: Clear the wakeup event received from PA0.
 - PWR_WAKEUP_FLAG2: Clear the wakeup event received from PA2.
 - PWR_WAKEUP_FLAG3: Clear the wakeup event received from PC1.
 - PWR_WAKEUP_FLAG4: Clear the wakeup event received from PC13.
 - PWR_WAKEUP_FLAG5: Clear the wakeup event received from PI8.
 - PWR_WAKEUP_FLAG6: Clear the wakeup event received from PI11.

Return values

- **HAL:** status.

`HAL_PWREx_WAKEUP_PIN_IRQHandler`

Function name

`void HAL_PWREx_WAKEUP_PIN_IRQHandler (void)`

Function description

This function handles the PWR WAKEUP PIN interrupt request.

Return values

- **None:**

Notes

- This API should be called under the `WAKEUP_PIN_IRQHandler()`.

`HAL_PWREx_WKUP1_Callback`

Function name

`void HAL_PWREx_WKUP1_Callback (void)`

Function description

PWR WKUP1 interrupt callback.

Return values

- None:

HAL_PWREx_WKUP2_Callback**Function name****void HAL_PWREx_WKUP2_Callback (void)****Function description**

PWR WKUP2 interrupt callback.

Return values

- None:

HAL_PWREx_WKUP3_Callback**Function name****void HAL_PWREx_WKUP3_Callback (void)****Function description**

PWR WKUP3 interrupt callback.

Return values

- None:

HAL_PWREx_WKUP4_Callback**Function name****void HAL_PWREx_WKUP4_Callback (void)****Function description**

PWR WKUP4 interrupt callback.

Return values

- None:

HAL_PWREx_WKUP5_Callback**Function name****void HAL_PWREx_WKUP5_Callback (void)****Function description**

PWR WKUP5 interrupt callback.

Return values

- None:

HAL_PWREx_WKUP6_Callback**Function name****void HAL_PWREx_WKUP6_Callback (void)****Function description**

PWR WKUP6 interrupt callback.

Return values

- None:

HAL_PWREx_EnableBkUpReg**Function name****HAL_StatusTypeDef HAL_PWREx_EnableBkUpReg (void)****Function description**

Enable the Backup Regulator.

Return values

- **HAL:** status

HAL_PWREx_DisableBkUpReg**Function name****HAL_StatusTypeDef HAL_PWREx_DisableBkUpReg (void)****Function description**

Disable the Backup Regulator.

Return values

- **HAL:** status

HAL_PWREx_EnableUSBReg**Function name****HAL_StatusTypeDef HAL_PWREx_EnableUSBReg (void)****Function description**

Enable the USB Regulator.

Return values

- **HAL:** status

HAL_PWREx_DisableUSBReg**Function name****HAL_StatusTypeDef HAL_PWREx_DisableUSBReg (void)****Function description**

Disable the USB Regulator.

Return values

- **HAL:** status

HAL_PWREx_EnableUSBVoltageDetector**Function name****void HAL_PWREx_EnableUSBVoltageDetector (void)****Function description**

Enable the USB voltage level detector.

Return values

- **None:**

HAL_PWREx_DisableUSBVoltageDetector

Function name

```
void HAL_PWREx_DisableUSBVoltageDetector (void )
```

Function description

Disable the USB voltage level detector.

Return values

- **None:**

HAL_PWREx_EnableBatteryCharging

Function name

```
void HAL_PWREx_EnableBatteryCharging (uint32_t ResistorValue)
```

Function description

Enable the Battery charging.

Parameters

- **ResistorValue:** Specifies the charging resistor. This parameter can be one of the following values:
 - PWR_BATTERY_CHARGING_RESISTOR_5: 5 KOhm resistor.
 - PWR_BATTERY_CHARGING_RESISTOR_1_5: 1.5 KOhm resistor.

Return values

- **None:**

HAL_PWREx_DisableBatteryCharging

Function name

```
void HAL_PWREx_DisableBatteryCharging (void )
```

Function description

Disable the Battery charging.

Return values

- **None:**

HAL_PWREx_EnableMonitoring

Function name

```
void HAL_PWREx_EnableMonitoring (void )
```

Function description

Enable the VBAT and temperature monitoring.

Return values

- **HAL:** status

HAL_PWREx_DisableMonitoring

Function name

```
void HAL_PWREx_DisableMonitoring (void )
```

Function description

Disable the VBAT and temperature monitoring.

Return values

- **HAL:** status

HAL_PWREx_GetTemperatureLevel

Function name

uint32_t HAL_PWREx_GetTemperatureLevel (void)

Function description

Indicate whether the junction temperature is between, above or below the threshold.

Return values

- **Temperature:** level.

HAL_PWREx_GetVBATLevel

Function name

uint32_t HAL_PWREx_GetVBATLevel (void)

Function description

Indicate whether the Battery voltage level is between, above or below the threshold.

Return values

- **VBAT:** level.

HAL_PWREx_ConfigAVD

Function name

void HAL_PWREx_ConfigAVD (PWREx_AVDTypeDef * sConfigAVD)

Function description

Configure the analog voltage threshold detected by the Analog Voltage Detector(AVD).

Parameters

- **sConfigAVD:** pointer to an PWR_AVDTypeDef structure that contains the configuration information for the AVD.

Return values

- **None:**

Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

HAL_PWREx_EnableAVD

Function name

void HAL_PWREx_EnableAVD (void)

Function description

Enable the Analog Voltage Detector(AVD).

Return values

- **None:**

HAL_PWREx_DisableAVD

Function name

```
void HAL_PWREx_DisableAVD (void )
```

Function description

Disable the Analog Voltage Detector(AVD).

Return values

- **None:**

HAL_PWREx_PVD_AVD_IRQHandler

Function name

```
void HAL_PWREx_PVD_AVD_IRQHandler (void )
```

Function description

This function handles the PWR PVD/AVD interrupt request.

Return values

- **None:**

Notes

- This API should be called under the PVD_AVD_IRQHandler().

HAL_PWREx_AVDCallback

Function name

```
void HAL_PWREx_AVDCallback (void )
```

Function description

PWR AVD interrupt callback.

Return values

- **None:**

53.3 PWREx Firmware driver defines

53.3.1 PWREx

PWREx AVD detection level

PWR_AVDEVEL_0

PWR_AVDEVEL_1

PWR_AVDEVEL_2

PWR_AVDEVEL_3

PWREx AVD EXTI Line 16

PWR_EXTI_LINE_AVD

External interrupt line 16 Connected to the AVD EXTI Line

PWREx AVD Mode

PWR_AVD_MODE_NORMAL

Basic mode is used

PWR_AVD_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_AVD_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_AVD_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_AVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_AVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_AVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR Extended AVD Mode Mask**AVD_MODE_IT****AVD_MODE_EVT****AVD_RISING_EDGE****AVD_FALLING_EDGE****AVD_RISING_FALLING_EDGE*****PWREx D3 Domain State*****PWR_D3_DOMAIN_STOP****PWR_D3_DOMAIN_RUN*****PWREx Domains definition*****PWR_D1_DOMAIN****PWR_D2_DOMAIN****PWR_D3_DOMAIN*****PWREx Domain Flags definition*****PWR_CPU_FLAGS*****PWREx Exported Macro*****_HAL_PWR_AVD_EXTI_ENABLE_IT****Description:**

- Enable the AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_IT

Description:

- Disable the AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_EVENT

Description:

- Enable event on AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_EVENT

Description:

- Disable event on AVD EXTI Line 16.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the AVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the AVD Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the AVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the AVD Extended Interrupt Falling Trigger.

Return value:

- None.

__HAL_PWR_AVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- AVD EXTI line configuration: set rising & falling edge trigger.

Return value:

- None.

[__HAL_PWR_AVD_EXTI_DISABLE_RISING_FALLING_EDGE](#)

Description:

- Disable the AVD Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

[__HAL_PWR_AVD_EXTI_GET_FLAG](#)

Description:

- Check whether the specified AVD EXTI interrupt flag is set or not.

Return value:

- EXTI: AVD Line Status.

[__HAL_PWR_AVD_EXTI_CLEAR_FLAG](#)

Description:

- Clear the AVD EXTI flag.

Return value:

- None.

PWREx Private macros to check input parameters

[IS_PWR_SUPPLY](#)

[IS_PWR_STOP_MODE_REGULATOR_VOLTAGE](#)

[IS_PWR_DOMAIN](#)

[IS_D3_STATE](#)

[IS_PWR_WAKEUP_PIN](#)

[IS_PWR_WAKEUP_PIN_POLARITY](#)

[IS_PWR_WAKEUP_PIN_PULL](#)

[IS_PWR_WAKEUP_FLAG](#)

[IS_PWR_AVD_LEVEL](#)

[IS_PWR_AVD_MODE](#)

[IS_PWR_BATTERY_RESISTOR_SELECT](#)

PWREx Pin Polarity configuration

[PWR_PIN_POLARITY_HIGH](#)

[PWR_PIN_POLARITY_LOW](#)

PWREx Pin Pull configuration

[PWR_PIN_NO_PULL](#)

PWR_PIN_PULL_UP

PWR_PIN_PULL_DOWN

PWREx Regulator Voltage Scale

PWR_REGULATOR_SVOS_SCALE5

PWR_REGULATOR_SVOS_SCALE4

PWR_REGULATOR_SVOS_SCALE3

PWR Extended Flag Setting Time Out Value

PWR_FLAG_SETTING_DELAY_US

PWREx Supply configuration

PWR_LDO_SUPPLY

PWR_EXTERNAL_SOURCE_SUPPLY

PWR_SUPPLY_CONFIG_MASK

PWREx Temperature Thresholds

PWR_TEMP_BETWEEN_HIGH_LOW_THRESHOLD

PWR_TEMP_BELOW_LOW_THRESHOLD

PWR_TEMP_ABOVE_HIGH_THRESHOLD

PWR battery charging resistor selection

PWR_BATTERY_CHARGING_RESISTOR_5

VBAT charging through a 5 kOhms resistor

PWR_BATTERY_CHARGING_RESISTOR_1_5

VBAT charging through a 1.5 kOhms resistor

PWREx VBAT Thresholds

PWR_VBAT_BETWEEN_HIGH_LOW_THRESHOLD

PWR_VBAT_BELOW_LOW_THRESHOLD

PWR_VBAT_ABOVE_HIGH_THRESHOLD

PWREx Wake-Up Pins

PWR_WAKEUP_PIN6

PWR_WAKEUP_PIN5

PWR_WAKEUP_PIN4

PWR_WAKEUP_PIN3

PWR_WAKEUP_PIN2

PWR_WAKEUP_PIN1
PWR_WAKEUP_PIN6_HIGH
PWR_WAKEUP_PIN5_HIGH
PWR_WAKEUP_PIN4_HIGH
PWR_WAKEUP_PIN3_HIGH
PWR_WAKEUP_PIN2_HIGH
PWR_WAKEUP_PIN1_HIGH
PWR_WAKEUP_PIN6_LOW
PWR_WAKEUP_PIN5_LOW
PWR_WAKEUP_PIN4_LOW
PWR_WAKEUP_PIN3_LOW
PWR_WAKEUP_PIN2_LOW
PWR_WAKEUP_PIN1_LOW
PWR_EXTI_WAKEUP_PINS_MASK
PWREx Wakeup Pins Flags.

PWR_WAKEUP_FLAG1
Wakeup event on pin 1

PWR_WAKEUP_FLAG2
Wakeup event on pin 2

PWR_WAKEUP_FLAG3
Wakeup event on pin 3

PWR_WAKEUP_FLAG4
Wakeup event on pin 4

PWR_WAKEUP_FLAG5
Wakeup event on pin 5

PWR_WAKEUP_FLAG6
Wakeup event on pin 6

54 HAL QSPI Generic Driver

54.1 QSPI Firmware driver registers structures

54.1.1 QSPI_InitTypeDef

Data Fields

- *uint32_t ClockPrescaler*
- *uint32_t FifoThreshold*
- *uint32_t SampleShifting*
- *uint32_t FlashSize*
- *uint32_t ChipSelectHighTime*
- *uint32_t ClockMode*
- *uint32_t FlashID*
- *uint32_t DualFlash*

Field Documentation

- *uint32_t QSPI_InitTypeDef::ClockPrescaler*
- *uint32_t QSPI_InitTypeDef::FifoThreshold*
- *uint32_t QSPI_InitTypeDef::SampleShifting*
- *uint32_t QSPI_InitTypeDef::FlashSize*
- *uint32_t QSPI_InitTypeDef::ChipSelectHighTime*
- *uint32_t QSPI_InitTypeDef::ClockMode*
- *uint32_t QSPI_InitTypeDef::FlashID*
- *uint32_t QSPI_InitTypeDef::DualFlash*

54.1.2 QSPI_HandleTypeDef

Data Fields

- *QUADSPI_TypeDef * Instance*
- *QSPI_InitTypeDef Init*
- *uint8_t * pTxBuffPtr*
- *_IO uint32_t TxXferSize*
- *_IO uint32_t TxXferCount*
- *uint8_t * pRxBuffPtr*
- *_IO uint32_t RxXferSize*
- *_IO uint32_t RxXferCount*
- *MDMA_HandleTypeDef * hmdma*
- *_IO HAL_LockTypeDef Lock*
- *_IO HAL_QSPI_StateTypeDef State*
- *_IO uint32_t ErrorCode*
- *uint32_t Timeout*

Field Documentation

- `QUADSPI_TypeDef* QSPI_HandleTypeDef::Instance`
- `QSPI_InitTypeDef QSPI_HandleTypeDef::Init`
- `uint8_t* QSPI_HandleTypeDef::pTxBuffPtr`
- `__IO uint32_t QSPI_HandleTypeDef::TxXferSize`
- `__IO uint32_t QSPI_HandleTypeDef::TxXferCount`
- `uint8_t* QSPI_HandleTypeDef::pRxBuffPtr`
- `__IO uint32_t QSPI_HandleTypeDef::RxXferSize`
- `__IO uint32_t QSPI_HandleTypeDef::RxXferCount`
- `MDMA_HandleTypeDef* QSPI_HandleTypeDef::hmdma`
- `__IO HAL_LockTypeDef QSPI_HandleTypeDef::Lock`
- `__IO HAL_QSPI_StateTypeDef QSPI_HandleTypeDef::State`
- `__IO uint32_t QSPI_HandleTypeDef::ErrorCode`
- `uint32_t QSPI_HandleTypeDef::Timeout`

54.1.3 QSPI_CommandTypeDef

Data Fields

- `uint32_t Instruction`
- `uint32_t Address`
- `uint32_t AlternateBytes`
- `uint32_t AddressSize`
- `uint32_t AlternateBytesSize`
- `uint32_t DummyCycles`
- `uint32_t InstructionMode`
- `uint32_t AddressMode`
- `uint32_t AlternateByteMode`
- `uint32_t DataMode`
- `uint32_t NbData`
- `uint32_t DdrMode`
- `uint32_t DdrHoldHalfCycle`
- `uint32_t SIOOMode`

Field Documentation

- `uint32_t QSPI_CommandTypeDef::Instruction`
- `uint32_t QSPI_CommandTypeDef::Address`
- `uint32_t QSPI_CommandTypeDef::AlternateBytes`
- `uint32_t QSPI_CommandTypeDef::AddressSize`
- `uint32_t QSPI_CommandTypeDef::AlternateBytesSize`

- `uint32_t QSPI_CommandTypeDef::DummyCycles`
- `uint32_t QSPI_CommandTypeDef::InstructionMode`
- `uint32_t QSPI_CommandTypeDef::AddressMode`
- `uint32_t QSPI_CommandTypeDef::AlternateByteMode`
- `uint32_t QSPI_CommandTypeDef::DataMode`
- `uint32_t QSPI_CommandTypeDef::NbData`
- `uint32_t QSPI_CommandTypeDef::DdrMode`
- `uint32_t QSPI_CommandTypeDef::DdrHoldHalfCycle`
- `uint32_t QSPI_CommandTypeDef::SIOOMode`

54.1.4 QSPI_AutoPollingTypeDef

Data Fields

- `uint32_t Match`
- `uint32_t Mask`
- `uint32_t Interval`
- `uint32_t StatusBytesSize`
- `uint32_t MatchMode`
- `uint32_t AutomaticStop`

Field Documentation

- `uint32_t QSPI_AutoPollingTypeDef::Match`
- `uint32_t QSPI_AutoPollingTypeDef::Mask`
- `uint32_t QSPI_AutoPollingTypeDef::Interval`
- `uint32_t QSPI_AutoPollingTypeDef::StatusBytesSize`
- `uint32_t QSPI_AutoPollingTypeDef::MatchMode`
- `uint32_t QSPI_AutoPollingTypeDef::AutomaticStop`

54.1.5 QSPI_MemoryMappedTypeDef

Data Fields

- `uint32_t TimeOutPeriod`
- `uint32_t TimeOutActivation`

Field Documentation

- `uint32_t QSPI_MemoryMappedTypeDef::TimeOutPeriod`
- `uint32_t QSPI_MemoryMappedTypeDef::TimeOutActivation`

54.2 QSPI Firmware driver API description

54.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_QSPI_MspInit() :
 - Enable QuadSPI clock interface with __HAL_RCC_QSPI_CLK_ENABLE().
 - Reset QuadSPI IP with __HAL_RCC_QSPI_FORCE_RESET() and __HAL_RCC_QSPI_RELEASE_RESET().
 - Enable the clocks for the QuadSPI GPIOs with __HAL_RCC_GPIOx_CLK_ENABLE().
 - Configure these QuadSPI pins in alternate mode using HAL_GPIO_Init().
 - If interrupt mode is used, enable and configure QuadSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
 - If DMA mode is used, enable the clocks for the QuadSPI MDMA with __HAL_RCC_MDMA_CLK_ENABLE(), configure MDMA with HAL_MDMA_Init(), link it with QuadSPI handle using __HAL_LINKDMA(), enable and configure MDMA global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the flash size, the clock prescaler, the fifo threshold, the clock mode, the sample shifting and the CS high time using the HAL_QSPI_Init() function.

Indirect functional mode

1. Configure the command sequence using the HAL_QSPI_Command() or HAL_QSPI_Command_IT() functions :
 - Instruction phase : the mode used and if present the instruction opcode.
 - Address phase : the mode used and if present the size and the address value.
 - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
 - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
 - Data phase : the mode used and if present the number of bytes.
 - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
 - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
2. If no data is required for the command, it is sent directly to the memory :
 - In polling mode, the output of the function is done when the transfer is complete.
 - In interrupt mode, HAL_QSPI_CmdCpltCallback() will be called when the transfer is complete.
3. For the indirect write mode, use HAL_QSPI_Transmit(), HAL_QSPI_Transmit_DMA() or HAL_QSPI_Transmit_IT() after the command configuration :
 - In polling mode, the output of the function is done when the transfer is complete.
 - In interrupt mode, HAL_QSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_QSPI_TxCpltCallback() will be called when the transfer is complete.
 - In DMA mode, HAL_QSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_QSPI_TxCpltCallback() will be called when the transfer is complete.
4. For the indirect read mode, use HAL_QSPI_Receive(), HAL_QSPI_Receive_DMA() or HAL_QSPI_Receive_IT() after the command configuration :
 - In polling mode, the output of the function is done when the transfer is complete.
 - In interrupt mode, HAL_QSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_QSPI_RxCpltCallback() will be called when the transfer is complete.
 - In DMA mode, HAL_QSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_QSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

1. Configure the command sequence and the auto-polling functional mode using the HAL_QSPI_AutoPolling() or HAL_QSPI_AutoPolling_IT() functions :
 - Instruction phase : the mode used and if present the instruction opcode.
 - Address phase : the mode used and if present the size and the address value.

- Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
 - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
 - Data phase : the mode used.
 - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
 - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
 - The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.
2. After the configuration :
 - In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
 - In interrupt mode, HAL_QSPI_StatusMatchCallback() will be called each time the status match is reached.

Memory-mapped functional mode

1. Configure the command sequence and the memory-mapped functional mode using the HAL_QSPI_MemoryMapped() functions :
 - Instruction phase : the mode used and if present the instruction opcode.
 - Address phase : the mode used and the size.
 - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
 - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
 - Data phase : the mode used.
 - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
 - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
 - The timeout activation and the timeout period.
2. After the configuration, the QuadSPI will be used as soon as an access on the AHB is done on the address range. HAL_QSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

1. HAL_QSPI_GetError() function gives the error raised during the last operation.
2. HAL_QSPI_Abort() and HAL_QSPI_AbortIT() functions aborts any on-going operation and flushes the fifo :
 - In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
 - In interrupt mode, HAL_QSPI_AbortCpltCallback() will be called when the transfer complete bi is set.

Control functions

1. HAL_QSPI_GetState() function gives the current state of the HAL QuadSPI driver.
2. HAL_QSPI_SetTimeout() function configures the timeout value used in the driver.
3. HAL_QSPI_SetFifoThreshold() function configures the threshold on the Fifo of the QSPI IP.
4. HAL_QSPI_GetFifoThreshold() function gives the current of the Fifo's threshold

Workarounds linked to Silicon Limitation

1. Workarounds Implemented inside HAL Driver
 - Extra data written in the FIFO at the end of a read transfer

54.2.2

Initialization and Configuration functions

This subsection provides a set of functions allowing to :

- Initialize the QuadSPI.
- De-initialize the QuadSPI.

This section contains the following APIs:

- [*HAL_QSPI_Init*](#)
- [*HAL_QSPI_DelInit*](#)
- [*HAL_QSPI_MspInit*](#)
- [*HAL_QSPI_MspDelInit*](#)

54.2.3 IO operation functions

This subsection provides a set of functions allowing to :

- Handle the interrupts.
- Handle the command sequence.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- [*HAL_QSPI_IRQHandler*](#)
- [*HAL_QSPI_Command*](#)
- [*HAL_QSPI_Command_IT*](#)
- [*HAL_QSPI_Transmit*](#)
- [*HAL_QSPI_Receive*](#)
- [*HAL_QSPI_Transmit_IT*](#)
- [*HAL_QSPI_Receive_IT*](#)
- [*HAL_QSPI_Transmit_DMA*](#)
- [*HAL_QSPI_Receive_DMA*](#)
- [*HAL_QSPI_AutoPolling*](#)
- [*HAL_QSPI_AutoPolling_IT*](#)
- [*HAL_QSPI_MemoryMapped*](#)
- [*HAL_QSPI_ErrorCallback*](#)
- [*HAL_QSPI_AbortCpltCallback*](#)
- [*HAL_QSPI_CmdCpltCallback*](#)
- [*HAL_QSPI_RxCpltCallback*](#)
- [*HAL_QSPI_TxCpltCallback*](#)
- [*HAL_QSPI_RxHalfCpltCallback*](#)
- [*HAL_QSPI_TxHalfCpltCallback*](#)
- [*HAL_QSPI_FifoThresholdCallback*](#)
- [*HAL_QSPI_StatusMatchCallback*](#)
- [*HAL_QSPI_TimeOutCallback*](#)

54.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to :

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.

This section contains the following APIs:

- [*HAL_QSPI_GetState*](#)
- [*HAL_QSPI_GetError*](#)
- [*HAL_QSPI_Abort*](#)
- [*HAL_QSPI_Abort_IT*](#)

- *HAL_QSPI_SetTimeout*
- *HAL_QSPI_SetFifoThreshold*
- *HAL_QSPI_GetFifoThreshold*

54.2.5 Detailed description of functions

HAL_QSPI_Init

Function name

HAL_StatusTypeDef HAL_QSPI_Init (QSPI_HandleTypeDef * hqspi)

Function description

Initialize the QSPI mode according to the specified parameters in the QSPI_InitTypeDef and initialize the associated handle.

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** status

HAL_QSPI_DeInit

Function name

HAL_StatusTypeDef HAL_QSPI_DeInit (QSPI_HandleTypeDef * hqspi)

Function description

De-Initialize the QSPI peripheral.

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** status

HAL_QSPI_MspInit

Function name

void HAL_QSPI_MspInit (QSPI_HandleTypeDef * hqspi)

Function description

Initialize the QSPI MSP.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_MspDeInit

Function name

void HAL_QSPI_MspDeInit (QSPI_HandleTypeDef * hqspi)

Function description

Deinitialize the QSPI MSP.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_IRQHandler**Function name**

void HAL_QSPI_IRQHandler (QSPI_HandleTypeDef * hqspi)

Function description

Handle QSPI interrupt request.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_Command**Function name**

HAL_StatusTypeDef HAL_QSPI_Command (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, uint32_t Timeout)

Function description

Set the command configuration.

Parameters

- **hqspi:** QSPI handle
- **cmd:** : structure that contains the command configuration information
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_QSPI_Transmit**Function name**

HAL_StatusTypeDef HAL_QSPI_Transmit (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_QSPI_Receive

Function name

```
HAL_StatusTypeDef HAL_QSPI_Receive (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t Timeout)
```

Function description

Receive an amount of data in blocking mode.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_QSPI_Command_IT

Function name

```
HAL_StatusTypeDef HAL_QSPI_Command_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd)
```

Function description

Set the command configuration in interrupt mode.

Parameters

- **hqspi:** QSPI handle
- **cmd:** : structure that contains the command configuration information

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_QSPI_Transmit_IT

Function name

```
HAL_StatusTypeDef HAL_QSPI_Transmit_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)
```

Function description

Send an amount of data in non-blocking mode with interrupt.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_QSPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Receive_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_QSPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_QSPI_Transmit_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with DMA.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode
- If MDMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If MDMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_QSPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_QSPI_Receive_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hqspi:** QSPI handle
- **pData:** pointer to data buffer.

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_QSPI_AutoPolling

Function name

HAL_StatusTypeDef HAL_QSPI_AutoPolling (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)

Function description

Configure the QSPI Automatic Polling Mode in blocking mode.

Parameters

- **hqspi:** QSPI handle
- **cmd:** structure that contains the command configuration information.
- **cfg:** structure that contains the polling configuration information.
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_QSPI_AutoPolling_IT

Function name

HAL_StatusTypeDef HAL_QSPI_AutoPolling_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg)

Function description

Configure the QSPI Automatic Polling Mode in non-blocking mode.

Parameters

- **hqspi:** QSPI handle
- **cmd:** structure that contains the command configuration information.
- **cfg:** structure that contains the polling configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_QSPI_MemoryMapped

Function name

HAL_StatusTypeDef HAL_QSPI_MemoryMapped (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_MemoryMappedTypeDef * cfg)

Function description

Configure the Memory Mapped mode.

Parameters

- **hqspi:** QSPI handle
- **cmd:** structure that contains the command configuration information.
- **cfg:** structure that contains the memory mapped configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Memory mapped Mode

HAL_QSPI_ErrorCallback

Function name

void HAL_QSPI_ErrorCallback (QSPI_HandleTypeDef * hqspi)

Function description

Transfer Error callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_AbortCpltCallback

Function name

void HAL_QSPI_AbortCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Abort completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_FifoThresholdCallback

Function name

void HAL_QSPI_FifoThresholdCallback (QSPI_HandleTypeDef * hqspi)

Function description

FIFO Threshold callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_CmdCpltCallback

Function name

void HAL_QSPI_CmdCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Command completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_RxCpltCallback

Function name

void HAL_QSPI_RxCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Rx Transfer completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_TxCpltCallback

Function name

void HAL_QSPI_TxCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Tx Transfer completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_RxHalfCpltCallback

Function name

void HAL_QSPI_RxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_TxHalfCpltCallback

Function name

void HAL_QSPI_TxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_StatusMatchCallback

Function name

void HAL_QSPI_StatusMatchCallback (QSPI_HandleTypeDef * hqspi)

Function description

Status Match callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_TimeOutCallback

Function name

void HAL_QSPI_TimeOutCallback (QSPI_HandleTypeDef * hqspi)

Function description

Timeout callback.

Parameters

- **hqspi:** QSPI handle

Return values

- **None:**

HAL_QSPI_GetState

Function name

HAL_QSPI_StateTypeDef HAL_QSPI_GetState (QSPI_HandleTypeDef * hqspi)

Function description

Return the QSPI handle state.

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** state

HAL_QSPI_GetError

Function name

uint32_t HAL_QSPI_GetError (QSPI_HandleTypeDef * hqspi)

Function description

Return the QSPI error code.

Parameters

- **hqspi:** QSPI handle

Return values

- **QSPI:** Error Code

HAL_QSPI_Abort

Function name

HAL_StatusTypeDef HAL_QSPI_Abort (QSPI_HandleTypeDef * hqspi)

Function description

Abort the current transmission.

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** status

HAL_QSPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_QSPI_Abort_IT (QSPI_HandleTypeDef * hqspi)

Function description

Abort the current transmission (non-blocking function)

Parameters

- **hqspi:** QSPI handle

Return values

- **HAL:** status

HAL_QSPI_SetTimeout

Function name

void HAL_QSPI_SetTimeout (QSPI_HandleTypeDef * hqspi, uint32_t Timeout)

Function description

Set QSPI timeout.

Parameters

- **hqspi:** QSPI handle.
- **Timeout:** Timeout for the QSPI memory access.

Return values

- **None:**

HAL_QSPI_SetFifoThreshold

Function name

HAL_StatusTypeDef HAL_QSPI_SetFifoThreshold (QSPI_HandleTypeDef * hqspi, uint32_t Threshold)

Function description

Set QSPI Fifo threshold.

Parameters

- **hqspi:** QSPI handle.
- **Threshold:** Threshold of the Fifo (value between 1 and 16).

Return values

- **HAL:** status

HAL_QSPI_GetFifoThreshold

Function name

uint32_t HAL_QSPI_GetFifoThreshold (QSPI_HandleTypeDef * hqspi)

Function description

Get QSPI Fifo threshold.

Parameters

- **hqspi:** QSPI handle.

Return values

- **Fifo:** threshold (value between 1 and 16)

54.3 QSPI Firmware driver defines

54.3.1 QSPI

QSPI Address Mode

QSPI_ADDRESS_NONE

No address

QSPI_ADDRESS_1_LINE

Address on a single line

QSPI_ADDRESS_2_LINES

Address on two lines

QSPI_ADDRESS_4_LINES

Address on four lines

QSPI Address Size

QSPI_ADDRESS_8_BITS

8-bit address

QSPI_ADDRESS_16_BITS

16-bit address

QSPI_ADDRESS_24_BITS

24-bit address

QSPI_ADDRESS_32_BITS

32-bit address

QSPI Alternate Bytes Mode**QSPI_ALTERNATE_BYTES_NONE**

No alternate bytes

QSPI_ALTERNATE_BYTES_1_LINE

Alternate bytes on a single line

QSPI_ALTERNATE_BYTES_2_LINES

Alternate bytes on two lines

QSPI_ALTERNATE_BYTES_4_LINES

Alternate bytes on four lines

QSPI Alternate Bytes Size**QSPI_ALTERNATE_BYTES_8_BITS**

8-bit alternate bytes

QSPI_ALTERNATE_BYTES_16_BITS

16-bit alternate bytes

QSPI_ALTERNATE_BYTES_24_BITS

24-bit alternate bytes

QSPI_ALTERNATE_BYTES_32_BITS

32-bit alternate bytes

QSPI Automatic Stop**QSPI_AUTOMATIC_STOP_DISABLE**

AutoPolling stops only with abort or QSPI disabling

QSPI_AUTOMATIC_STOP_ENABLE

AutoPolling stops as soon as there is a match

QSPI ChipSelect High Time**QSPI_CS_HIGH_TIME_1_CYCLE**

nCS stay high for at least 1 clock cycle between commands

QSPI_CS_HIGH_TIME_2_CYCLE

nCS stay high for at least 2 clock cycles between commands

QSPI_CS_HIGH_TIME_3_CYCLE

nCS stay high for at least 3 clock cycles between commands

QSPI_CS_HIGH_TIME_4_CYCLE

nCS stay high for at least 4 clock cycles between commands

QSPI_CS_HIGH_TIME_5_CYCLE

nCS stay high for at least 5 clock cycles between commands

QSPI_CS_HIGH_TIME_6_CYCLE

nCS stay high for at least 6 clock cycles between commands

QSPI_CS_HIGH_TIME_7_CYCLE

nCS stay high for at least 7 clock cycles between commands

QSPI_CS_HIGH_TIME_8_CYCLE

nCS stay high for at least 8 clock cycles between commands

QSPI Clock Mode**QSPI_CLOCK_MODE_0**

Clk stays low while nCS is released

QSPI_CLOCK_MODE_3

Clk goes high while nCS is released

QSPI Data Mode**QSPI_DATA_NONE**

No data

QSPI_DATA_1_LINE

Data on a single line

QSPI_DATA_2_LINES

Data on two lines

QSPI_DATA_4_LINES

Data on four lines

QSPI DDR Data Output Delay**QSPI_DDR_HHC_ANALOG_DELAY**

Delay the data output using analog delay in DDR mode

QSPI_DDR_HHC_HALF_CLK_DELAY

Delay the data output by 1/2 clock cycle in DDR mode

QSPI DDR Mode**QSPI_DDR_MODE_DISABLE**

Double data rate mode disabled

QSPI_DDR_MODE_ENABLE

Double data rate mode enabled

QSPI Dual Flash Mode**QSPI_DUALFLASH_ENABLE**

Dual-flash mode enabled

QSPI_DUALFLASH_DISABLE

Dual-flash mode disabled

QSPI Error Code**HAL_QSPI_ERROR_NONE**

No error

HAL_QSPI_ERROR_TIMEOUT

Timeout error

HAL_QSPI_ERROR_TRANSFER

Transfer error

HAL_QSPI_ERROR_DMA

DMA transfer error

HAL_QSPI_ERROR_INVALID_PARAM

Invalid parameters error

QSPI Exported Macros

__HAL_QSPI_RESET_HANDLE_STATE

Description:

- Reset QSPI handle state.

Parameters:

- __HANDLE__: QSPI handle.

Return value:

- None

__HAL_QSPI_ENABLE

Description:

- Enable the QSPI peripheral.

Parameters:

- __HANDLE__: specifies the QSPI Handle.

Return value:

- None

__HAL_QSPI_DISABLE

Description:

- Disable the QSPI peripheral.

Parameters:

- __HANDLE__: specifies the QSPI Handle.

Return value:

- None

__HAL_QSPI_ENABLE_IT

Description:

- Enable the specified QSPI interrupt.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI interrupt source to enable. This parameter can be one of the following values:
 - QSPI_IT_TO: QSPI Timeout interrupt
 - QSPI_IT_SM: QSPI Status match interrupt
 - QSPI_IT_FT: QSPI FIFO threshold interrupt
 - QSPI_IT_TC: QSPI Transfer complete interrupt
 - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- None

[__HAL_QSPI_DISABLE_IT](#)

Description:

- Disable the specified QSPI interrupt.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI interrupt source to disable. This parameter can be one of the following values:
 - QSPI_IT_TO: QSPI Timeout interrupt
 - QSPI_IT_SM: QSPI Status match interrupt
 - QSPI_IT_FT: QSPI FIFO threshold interrupt
 - QSPI_IT_TC: QSPI Transfer complete interrupt
 - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

- None

[__HAL_QSPI_GET_IT_SOURCE](#)

Description:

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

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __INTERRUPT__: specifies the QSPI interrupt source to check. This parameter can be one of the following values:
 - QSPI_IT_TO: QSPI Timeout interrupt
 - QSPI_IT_SM: QSPI Status match interrupt
 - QSPI_IT_FT: QSPI FIFO threshold interrupt
 - QSPI_IT_TC: QSPI Transfer complete interrupt
 - QSPI_IT_TE: QSPI Transfer error interrupt

Return value:

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

[__HAL_QSPI_GET_FLAG](#)

Description:

- Check whether the selected QSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __FLAG__: specifies the QSPI flag to check. This parameter can be one of the following values:
 - QSPI_FLAG_BUSY: QSPI Busy flag
 - QSPI_FLAG_TO: QSPI Timeout flag
 - QSPI_FLAG_SM: QSPI Status match flag
 - QSPI_FLAG_FT: QSPI FIFO threshold flag
 - QSPI_FLAG_TC: QSPI Transfer complete flag
 - QSPI_FLAG_TE: QSPI Transfer error flag

Return value:

- None

__HAL_QSPI_CLEAR_FLAG

Description:

- Clears the specified QSPI's flag status.

Parameters:

- __HANDLE__: specifies the QSPI Handle.
- __FLAG__: specifies the QSPI clear register flag that needs to be set This parameter can be one of the following values:
 - QSPI_FLAG_TO: QSPI Timeout flag
 - QSPI_FLAG_SM: QSPI Status match flag
 - QSPI_FLAG_TC: QSPI Transfer complete flag
 - QSPI_FLAG_TE: QSPI Transfer error flag

Return value:

- None

QSPI Flags

QSPI_FLAG_BUSY

Busy flag: operation is ongoing

QSPI_FLAG_TO

Timeout flag: timeout occurs in memory-mapped mode

QSPI_FLAG_SM

Status match flag: received data matches in autopolling mode

QSPI_FLAG_FT

Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete

QSPI_FLAG_TC

Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted

QSPI_FLAG_TE

Transfer error flag: invalid address is being accessed

QSPI Flash Select

QSPI_FLASH_ID_1

FLASH 1 selected

QSPI_FLASH_ID_2

FLASH 2 selected

QSPI Instruction Mode

QSPI_INSTRUCTION_NONE

No instruction

QSPI_INSTRUCTION_1_LINE

Instruction on a single line

QSPI_INSTRUCTION_2_LINES

Instruction on two lines

QSPI_INSTRUCTION_4_LINES

Instruction on four lines

QSPI Interrupts**QSPI_IT_TO**

Interrupt on the timeout flag

QSPI_IT_SM

Interrupt on the status match flag

QSPI_IT_FT

Interrupt on the fifo threshold flag

QSPI_IT_TC

Interrupt on the transfer complete flag

QSPI_IT_TE

Interrupt on the transfer error flag

QSPI Match Mode**QSPI_MATCH_MODE_AND**

AND match mode between unmasked bits

QSPI_MATCH_MODE_OR

OR match mode between unmasked bits

QSPI Sample Shifting**QSPI_SAMPLE_SHIFTING_NONE**

No clock cycle shift to sample data

QSPI_SAMPLE_SHIFTING_HALFCYCLE

1/2 clock cycle shift to sample data

QSPI Send Instruction Mode**QSPI_SIOO_INST_EVERY_CMD**

Send instruction on every transaction

QSPI_SIOO_INST_ONLY_FIRST_CMD

Send instruction only for the first command

QSPI Timeout Activation**QSPI_TIMEOUT_COUNTER_DISABLE**

Timeout counter disabled, nCS remains active

QSPI_TIMEOUT_COUNTER_ENABLE

Timeout counter enabled, nCS released when timeout expires

QSPI Timeout definition**HAL_QPSI_TIMEOUT_DEFAULT_VALUE**

55 HAL RCC Generic Driver

55.1 RCC Firmware driver registers structures

55.1.1 RCC_PLLInitTypeDef

Data Fields

- *uint32_t PLLState*
- *uint32_t PLLSource*
- *uint32_t PLLM*
- *uint32_t PLLN*
- *uint32_t PLLP*
- *uint32_t PLLQ*
- *uint32_t PLLR*
- *uint32_t PLLRGE*
- *uint32_t PLLVCOSEL*
- *uint32_t PLLFRACN*

Field Documentation

- *uint32_t RCC_PLLInitTypeDef::PLLState*

The new state of the PLL. This parameter can be a value of **RCC PLL Config**

- *uint32_t RCC_PLLInitTypeDef::PLLSource*

RCC_PLLSource: PLL entry clock source. This parameter must be a value of **RCC PLL Clock Source**

- *uint32_t RCC_PLLInitTypeDef::PLLM*

PLLM: Division factor for PLL VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63

- *uint32_t RCC_PLLInitTypeDef::PLLN*

PLLN: Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512

- *uint32_t RCC_PLLInitTypeDef::PLLP*

PLLP: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowed

- *uint32_t RCC_PLLInitTypeDef::PLLQ*

PLLQ: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- *uint32_t RCC_PLLInitTypeDef::PLLR*

PLLR: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- *uint32_t RCC_PLLInitTypeDef::PLLRGE*

PLLRGE: PLL1 clock Input range This parameter must be a value of **RCC PLL1 VCI Range**

- *uint32_t RCC_PLLInitTypeDef::PLLVCOSEL*

PLLVCOSEL: PLL1 clock Output range This parameter must be a value of **RCC PLL1 VCO Range**

- *uint32_t RCC_PLLInitTypeDef::PLLFRACN*

PLLFRACN: Specifies Fractional Part Of The Multiplication Factor for PLL1 VCO It should be a value between 0 and 8191

55.1.2 RCC_OscInitTypeDef

Data Fields

- `uint32_t OscillatorType`
- `uint32_t HSEState`
- `uint32_t LSEState`
- `uint32_t HSIState`
- `uint32_t HSICalibrationValue`
- `uint32_t LSIState`
- `uint32_t HSI48State`
- `uint32_t CSISState`
- `uint32_t CSICalibrationValue`
- `RCC_PLLInitTypeDef PLL`

Field Documentation

- `uint32_t RCC_OscInitTypeDef::OscillatorType`

The oscillators to be configured. This parameter can be a value of **RCC Oscillator Type**

- `uint32_t RCC_OscInitTypeDef::HSEState`

The new state of the HSE. This parameter can be a value of **RCC HSE Config**

- `uint32_t RCC_OscInitTypeDef::LSEState`

The new state of the LSE. This parameter can be a value of **RCC LSE Config**

- `uint32_t RCC_OscInitTypeDef::HSIState`

The new state of the HSI. This parameter can be a value of **RCC HSI Config**

- `uint32_t RCC_OscInitTypeDef::HSICalibrationValue`

The calibration trimming value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x3F

- `uint32_t RCC_OscInitTypeDef::LSIState`

The new state of the LSI. This parameter can be a value of **RCC LSI Config**

- `uint32_t RCC_OscInitTypeDef::HSI48State`

The new state of the HSI48. This parameter can be a value of **RCC HSI48 Config**

- `uint32_t RCC_OscInitTypeDef::CSISState`

The new state of the CSI. This parameter can be a value of **RCC CSI Config**

- `uint32_t RCC_OscInitTypeDef::CSICalibrationValue`

The calibration trimming value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F

- `RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL`

PLL structure parameters

55.1.3 RCC_ClkInitTypeDef

Data Fields

- `uint32_t ClockType`
- `uint32_t SYSCLKSource`
- `uint32_t SYSCLKDivider`

- `uint32_t AHBCLKDivider`
- `uint32_t APB3CLKDivider`
- `uint32_t APB1CLKDivider`
- `uint32_t APB2CLKDivider`
- `uint32_t APB4CLKDivider`

Field Documentation

- `uint32_t RCC_ClkInitTypeDef::ClockType`

The clock to be configured. This parameter can be a value of **RCC System Clock Type**

- `uint32_t RCC_ClkInitTypeDef::SYSCLKSource`

The clock source (SYSCLKS) used as system clock. This parameter can be a value of **RCC System Clock Source**

- `uint32_t RCC_ClkInitTypeDef::SYSCLKDivider`

The system clock divider. This parameter can be a value of **RCC SYS Clock Source**

- `uint32_t RCC_ClkInitTypeDef::AHBCLKDivider`

The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of **RCC HCLK Clock Source**

- `uint32_t RCC_ClkInitTypeDef::APB3CLKDivider`

The APB3 clock (D1PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of **RCC APB3 Clock Source**

- `uint32_t RCC_ClkInitTypeDef::APB1CLKDivider`

The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of **RCC APB1 Clock Source**

- `uint32_t RCC_ClkInitTypeDef::APB2CLKDivider`

The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of **RCC APB2 Clock Source**

- `uint32_t RCC_ClkInitTypeDef::APB4CLKDivider`

The APB4 clock (D3PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of **RCC APB4 Clock Source**

55.2 RCC Firmware driver API description

55.2.1 RCC specific features

After reset the device is running from Internal High Speed oscillator (HSI 64MHz) with Flash 0 wait state, and all peripherals are off except internal SRAM, Flash, JTAG and PWR

- There is no pre-scaler on High speed (AHB) and Low speed (APB) buses; all peripherals mapped on these buses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in analogue mode , except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/ performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB buses pre-scalers
- Enable the clock for the peripheral(s) to be used

- Configure the clock kernel source(s) for peripherals which clocks are not derived from the System clock through :RCC_D1CCIPR,RCC_D2CCIP1R,RCC_D2CCIP2R and RCC_D3CCIPR registers

55.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Implemented Workaround:

- For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each `_HAL_RCC_PPP_CLK_ENABLE()` macro.

55.2.3 Initialization and de-initialization functions

This section provides functions allowing to configure the internal/external oscillators (HSE, HSI, LSE,CSI, LSI,HSI48, PLL, CSS and MCO) and the System buses clocks (SYSCLK, AHB3, AHB1 AHB2,AHB4,APB3, APB1L, APB1H, APB2, and APB4).

Internal/external clock and PLL configuration

- HSI (high-speed internal), 64 MHz factory-trimmed RC used directly or through the PLL as System clock source.
- CSI is a low-power RC oscillator which can be used directly as system clock, peripheral clock, or PLL input.But even with frequency calibration, is less accurate than an external crystal oscillator or ceramic resonator.
- LSI (low-speed internal), 32 KHz low consumption RC used as IWDG and/or RTC clock source.
- HSE (high-speed external), 4 to 48 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
- LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
- PLL , The RCC features three independent PLLs (clocked by HSI , HSE or CSI), featuring three different output clocks and able to work either in integer or Fractional mode.
 - A main PLL, PLL1, which is generally used to provide clocks to the CPU and to some peripherals.
 - Two dedicated PLLs, PLL2 and PLL3, which are used to generate the kernel clock for peripherals.
- CSS (Clock security system), once enabled and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M NMI (Non-Maskable Interrupt) exception vector.
- MCO1 (micro controller clock output), used to output HSI, LSE, HSE, PLL1(PLL1_Q) or HSI48 clock (through a configurable pre-scaler) on PA8 pin.
- MCO2 (micro controller clock output), used to output HSE, PLL2(PLL2_P), SYSCLK, LSI, CSI, or PLL1(PLL1_P) clock (through a configurable pre-scaler) on PC9 pin.

System, AHB and APB buses clocks configuration

- Several clock sources can be used to drive the System clock (SYSCLK): CSI,HSI, HSE and PLL. The AHB clock (HCLK) is derived from System core clock through configurable pre-scaler and used to clock the CPU, memory and peripherals mapped on AHB and APB bus of the 3 Domains (D1, D2, D3) through configurable pre-scalers and used to clock the peripherals mapped on these buses. You can use "`HAL_RCC_GetSysClockFreq()`" function to retrieve system clock frequency.

Note:

All the peripheral clocks are derived from the System clock (SYSCLK) except those with dual clock domain where kernel source clock could be selected through RCC_D1CCIPR,RCC_D2CCIP1R,RCC_D2CCIP2R and RCC_D3CCIPR registers.

This section contains the following APIs:

- [*HAL_RCC_DelInit*](#)
- [*HAL_RCC_OscConfig*](#)
- [*HAL_RCC_ClockConfig*](#)

55.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- [*HAL_RCC_MCOConfig*](#)
- [*HAL_RCC_EnableCSS*](#)
- [*HAL_RCC_GetSysClockFreq*](#)
- [*HAL_RCC_GetHCLKFreq*](#)
- [*HAL_RCC_GetPCLK1Freq*](#)
- [*HAL_RCC_GetPCLK2Freq*](#)
- [*HAL_RCC_GetOscConfig*](#)
- [*HAL_RCC_GetClockConfig*](#)
- [*HAL_RCC_NMI_IRQHandler*](#)
- [*HAL_RCC_CCSCallback*](#)

55.2.5 Detailed description of functions

HAL_RCC_DelInit

Function name

void HAL_RCC_DelInit (void)

Function description

Resets the RCC clock configuration to the default reset state.

Return values

- **None:**

Notes

- The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, PLL1, PLL2 and PLL3 OFFAHB, APB Bus pre-scaler set to 1.CSS, MCO1 and MCO2 OFFAII interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)

Function description

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters

- **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.

Return values

- **HAL:** status

Notes

- The PLL is not disabled when used as system clock.

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this function. User should request a transition to LSE Off first and then LSE On or LSE Bypass.
- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this function. User should request a transition to HSE Off first and then HSE On or HSE Bypass.

HAL_RCC_ClockConfig

Function name

```
HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)
```

Function description

Initializes the CPU, AHB and APB buses clocks according to the specified parameters in the RCC_ClkInitStruct.

Parameters

- **RCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.
- **FLatency:** FLASH Latency, this parameter depend on device selected

Return values

- **None:**

Notes

- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_InitTick() function called within this function
- The HSI is used (enabled by hardware) as system clock source after start-up from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after start-up delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready. You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.
- Depending on the device voltage range, the software has to set correctly D1CPRE[3:0] bits to ensure that Domain1 core clock not exceed the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")

HAL_RCC_MCOConfig

Function name

```
void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOsource, uint32_t RCC_MCODiv)
```

Function description

Selects the clock source to output on MCO1 pin(PA8) or on MCO2 pin(PC9).

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source. This parameter can be one of the following values:
 - RCC_MCO1: Clock source to output on MCO1 pin(PA8).
 - RCC_MCO2: Clock source to output on MCO2 pin(PC9).
- **RCC_MCOsource:** specifies the clock source to output. This parameter can be one of the following values:
 - RCC_MCO1SOURCE_HSI: HSI clock selected as MCO1 source
 - RCC_MCO1SOURCE_LSE: LSE clock selected as MCO1 source
 - RCC_MCO1SOURCE_HSE: HSE clock selected as MCO1 source
 - RCC_MCO1SOURCE_PLL1QCLK: PLL1Q clock selected as MCO1 source
 - RCC_MCO1SOURCE_HSI48: HSI48 (48MHZ) selected as MCO1 source

- RCC_MCO2SOURCE_SYSCLK: System clock (SYSCLK) selected as MCO2 source
- RCC_MCO2SOURCE_PLL2PCLK: PLL2P clock selected as MCO2 source
- RCC_MCO2SOURCE_HSE: HSE clock selected as MCO2 source
- RCC_MCO2SOURCE_PLLCLK: PLL1P clock selected as MCO2 source
- RCC_MCO2SOURCE_CSICLK: CSI clock selected as MCO2 source
- RCC_MCO2SOURCE_LSICLK: LSI clock selected as MCO2 source
- **RCC_MCODiv:** specifies the MCOx pre-scaler. This parameter can be one of the following values:
 - RCC_MCODIV_1 up to RCC_MCODIV_15 : divider applied to MCOx clock

Return values

- **None:**

Notes

- PA8/PC9 should be configured in alternate function mode.

HAL_RCC_EnableCSS

Function name

```
void HAL_RCC_EnableCSS (void )
```

Function description

Enables the Clock Security System.

Return values

- **None:**

Notes

- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M NMI (Non-Maskable Interrupt) exception vector.

HAL_RCC_DisableCSS

Function name

```
void HAL_RCC_DisableCSS (void )
```

Function description

HAL_RCC_GetSysClockFreq

Function name

```
uint32_t HAL_RCC_GetSysClockFreq (void )
```

Function description

Returns the SYSCLK frequency.

Return values

- **SYSCLK:** frequency

Notes

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is CSI, function returns values based on CSI_VALUE(*)
- If SYSCLK source is HSI, function returns values based on HSI_VALUE(**)
- If SYSCLK source is HSE, function returns values based on HSE_VALUE(***)

- If SYSCLK source is PLL, function returns values based on CSI_VALUE(*), HSI_VALUE(**) or HSE_VALUE(***) multiplied/divided by the PLL factors.
- (*) CSI_VALUE is a constant defined in stm32h7xx_hal_conf.h file (default value 4 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (***) HSI_VALUE is a constant defined in stm32h7xx_hal_conf.h file (default value 64 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (****) HSE_VALUE is a constant defined in stm32h7xx_hal_conf.h file (default value 25 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baud rate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetHCLKFreq

Function name

```
uint32_t HAL_RCC_GetHCLKFreq (void )
```

Function description

Returns the HCLK frequency.

Return values

- **HCLK:** frequency

Notes

- Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect.
- The SystemD2Clock CMSIS variable is used to store System domain2 Clock Frequency and updated within this function

HAL_RCC_GetPCLK1Freq

Function name

```
uint32_t HAL_RCC_GetPCLK1Freq (void )
```

Function description

Returns the PCLK1 frequency.

Return values

- **PCLK1:** frequency

Notes

- Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK2Freq

Function name

```
uint32_t HAL_RCC_GetPCLK2Freq (void )
```

Function description

Returns the PCLK2 frequency.

Return values

- **PCLK1:** frequency

Notes

- Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name

```
void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)
```

Function description

Configures the RCC_OscInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that will be configured.

Return values

- **None:**

HAL_RCC_GetClockConfig

Function name

```
void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)
```

Function description

Configures the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_ClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that will be configured.
- **pFLatency:** Pointer on the Flash Latency.

Return values

- **None:**

HAL_RCC_NMI_IRQHandler

Function name

```
void HAL_RCC_NMI_IRQHandler (void )
```

Function description

This function handles the RCC CSS interrupt request.

Return values

- **None:**

Notes

- This API should be called under the NMI_Handler().

HAL_RCC_CCSCallback

Function name

```
void HAL_RCC_CCSCallback (void )
```

Function description

RCC Clock Security System interrupt callback.

Return values

- **none:**

55.3 RCC Firmware driver defines

55.3.1 RCC

RCC APB1 Clock Source

`RCC_APB1_DIV1`

`RCC_APB1_DIV2`

`RCC_APB1_DIV4`

`RCC_APB1_DIV8`

`RCC_APB1_DIV16`

RCC APB2 Clock Source

`RCC_APB2_DIV1`

`RCC_APB2_DIV2`

`RCC_APB2_DIV4`

`RCC_APB2_DIV8`

`RCC_APB2_DIV16`

RCC APB3 Clock Source

`RCC_APB3_DIV1`

`RCC_APB3_DIV2`

`RCC_APB3_DIV4`

`RCC_APB3_DIV8`

`RCC_APB3_DIV16`

RCC APB4 Clock Source

`RCC_APB4_DIV1`

`RCC_APB4_DIV2`

`RCC_APB4_DIV4`

`RCC_APB4_DIV8`

`RCC_APB4_DIV16`

RCC CSI Config

`RCC_CSI_OFF`

`RCC_CSI_ON`

RCC_CSICALIBRATION_DEFAULT

RCC Exported Macros

[__HAL_RCC_MDMA_CLK_ENABLE](#)

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

[__HAL_RCC_DMA2D_CLK_ENABLE](#)

[__HAL_RCC_JPGDECEN_CLK_ENABLE](#)

[__HAL_RCC_FMC_CLK_ENABLE](#)

[__HAL_RCC_QSPI_CLK_ENABLE](#)

[__HAL_RCC_SDMMC1_CLK_ENABLE](#)

[__HAL_RCC_MDMA_CLK_DISABLE](#)

[__HAL_RCC_DMA2D_CLK_DISABLE](#)

[__HAL_RCC_JPGDECEN_CLK_DISABLE](#)

[__HAL_RCC_FMC_CLK_DISABLE](#)

[__HAL_RCC_QSPI_CLK_DISABLE](#)

[__HAL_RCC_SDMMC1_CLK_DISABLE](#)

[__HAL_RCC_DMA1_CLK_ENABLE](#)

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

[__HAL_RCC_DMA2_CLK_ENABLE](#)

[__HAL_RCC_ADC12_CLK_ENABLE](#)

[__HAL_RCC_ETH1MAC_CLK_ENABLE](#)

[__HAL_RCC_ETH1TX_CLK_ENABLE](#)

[__HAL_RCC_ETH1RX_CLK_ENABLE](#)

[__HAL_RCC_USB1_OTG_HS_CLK_ENABLE](#)

[__HAL_RCC_USB1_OTG_HS_ULPI_CLK_ENABLE](#)

[__HAL_RCC_USB2_OTG_FS_CLK_ENABLE](#)

[__HAL_RCC_USB2_OTG_FS_ULPI_CLK_ENABLE](#)

`_HAL_RCC_DMA1_CLK_DISABLE`
`_HAL_RCC_DMA2_CLK_DISABLE`
`_HAL_RCC_ADC12_CLK_DISABLE`
`_HAL_RCC_ETH1MAC_CLK_DISABLE`
`_HAL_RCC_ETH1TX_CLK_DISABLE`
`_HAL_RCC_ETH1RX_CLK_DISABLE`
`_HAL_RCC_USB1_OTG_HS_CLK_DISABLE`
`_HAL_RCC_USB1_OTG_HS_ULPI_CLK_DISABLE`
`_HAL_RCC_USB2_OTG_FS_CLK_DISABLE`
`_HAL_RCC_USB2_OTG_FS_ULPI_CLK_DISABLE`
`_HAL_RCC_DCMI_CLK_ENABLE`

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

`_HAL_RCC_CRYP_CLK_ENABLE`
`_HAL_RCC_HASH_CLK_ENABLE`
`_HAL_RCC RNG_CLK_ENABLE`
`_HAL_RCC_SDMMC2_CLK_ENABLE`
`_HAL_RCC_D2SRAM1_CLK_ENABLE`
`_HAL_RCC_D2SRAM2_CLK_ENABLE`
`_HAL_RCC_D2SRAM3_CLK_ENABLE`
`_HAL_RCC_DCMI_CLK_DISABLE`
`_HAL_RCC_CRYP_CLK_DISABLE`
`_HAL_RCC_HASH_CLK_DISABLE`
`_HAL_RCC RNG_CLK_DISABLE`
`_HAL_RCC_SDMMC2_CLK_DISABLE`
`_HAL_RCC_D2SRAM1_CLK_DISABLE`
`_HAL_RCC_D2SRAM2_CLK_DISABLE`
`_HAL_RCC_D2SRAM3_CLK_DISABLE`

[__HAL_RCC_GPIOA_CLK_ENABLE](#)**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

[__HAL_RCC_GPIOB_CLK_ENABLE](#)[__HAL_RCC_GPIOC_CLK_ENABLE](#)[__HAL_RCC_GPIOD_CLK_ENABLE](#)[__HAL_RCC_GPIOE_CLK_ENABLE](#)[__HAL_RCC_GPIOF_CLK_ENABLE](#)[__HAL_RCC_GPIOG_CLK_ENABLE](#)[__HAL_RCC_GPIOH_CLK_ENABLE](#)[__HAL_RCC_GPIOI_CLK_ENABLE](#)[__HAL_RCC_GPIOJ_CLK_ENABLE](#)[__HAL_RCC_GPIOK_CLK_ENABLE](#)[__HAL_RCC_CRC_CLK_ENABLE](#)[__HAL_RCC_BDMA_CLK_ENABLE](#)[__HAL_RCC_ADC3_CLK_ENABLE](#)[__HAL_RCC_HSEM_CLK_ENABLE](#)[__HAL_RCC_BKPRAM_CLK_ENABLE](#)[__HAL_RCC_GPIOA_CLK_DISABLE](#)[__HAL_RCC_GPIOB_CLK_DISABLE](#)[__HAL_RCC_GPIOC_CLK_DISABLE](#)[__HAL_RCC_GPIOD_CLK_DISABLE](#)[__HAL_RCC_GPIOE_CLK_DISABLE](#)[__HAL_RCC_GPIOF_CLK_DISABLE](#)[__HAL_RCC_GPIOG_CLK_DISABLE](#)[__HAL_RCC_GPIOH_CLK_DISABLE](#)[__HAL_RCC_GPIOI_CLK_DISABLE](#)[__HAL_RCC_GPIOJ_CLK_DISABLE](#)

`_HAL_RCC_GPIOK_CLK_DISABLE`
`_HAL_RCC_CRC_CLK_DISABLE`
`_HAL_RCC_BDMA_CLK_DISABLE`
`_HAL_RCC_ADC3_CLK_DISABLE`
`_HAL_RCC_HSEM_CLK_DISABLE`
`_HAL_RCC_BKPRAM_CLK_DISABLE`
`_HAL_RCC_LTDC_CLK_ENABLE`

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

`_HAL_RCC_WWDG1_CLK_ENABLE`
`_HAL_RCC_LTDC_CLK_DISABLE`
`_HAL_RCC_WWDG1_CLK_DISABLE`
`_HAL_RCC_TIM2_CLK_ENABLE`

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

`_HAL_RCC_TIM3_CLK_ENABLE`
`_HAL_RCC_TIM4_CLK_ENABLE`
`_HAL_RCC_TIM5_CLK_ENABLE`
`_HAL_RCC_TIM6_CLK_ENABLE`
`_HAL_RCC_TIM7_CLK_ENABLE`
`_HAL_RCC_TIM12_CLK_ENABLE`
`_HAL_RCC_TIM13_CLK_ENABLE`
`_HAL_RCC_TIM14_CLK_ENABLE`
`_HAL_RCC_LPTIM1_CLK_ENABLE`
`_HAL_RCC_SPI2_CLK_ENABLE`
`_HAL_RCC_SPI3_CLK_ENABLE`
`_HAL_RCC_SPDIFRX_CLK_ENABLE`
`_HAL_RCC_USART2_CLK_ENABLE`

```
_HAL_RCC_USART3_CLK_ENABLE
__HAL_RCC_UART4_CLK_ENABLE
__HAL_RCC_UART5_CLK_ENABLE
__HAL_RCC_I2C1_CLK_ENABLE
__HAL_RCC_I2C2_CLK_ENABLE
__HAL_RCC_I2C3_CLK_ENABLE
__HAL_RCC_CEC_CLK_ENABLE
__HAL_RCC_DAC12_CLK_ENABLE
__HAL_RCC_UART7_CLK_ENABLE
__HAL_RCC_UART8_CLK_ENABLE
__HAL_RCC_CRS_CLK_ENABLE
__HAL_RCC_SWPMI1_CLK_ENABLE
__HAL_RCC_OPAMP_CLK_ENABLE
__HAL_RCC_MDIOS_CLK_ENABLE
__HAL_RCC_FDCAN_CLK_ENABLE
__HAL_RCC_TIM2_CLK_DISABLE
__HAL_RCC_TIM3_CLK_DISABLE
__HAL_RCC_TIM4_CLK_DISABLE
__HAL_RCC_TIM5_CLK_DISABLE
__HAL_RCC_TIM6_CLK_DISABLE
__HAL_RCC_TIM7_CLK_DISABLE
__HAL_RCC_TIM12_CLK_DISABLE
__HAL_RCC_TIM13_CLK_DISABLE
__HAL_RCC_TIM14_CLK_DISABLE
__HAL_RCC_LPTIM1_CLK_DISABLE
__HAL_RCC_SPI2_CLK_DISABLE
__HAL_RCC_SPI3_CLK_DISABLE
__HAL_RCC_SPDIFRX_CLK_DISABLE
```

`_HAL_RCC_USART2_CLK_DISABLE`
`_HAL_RCC_USART3_CLK_DISABLE`
`_HAL_RCC_UART4_CLK_DISABLE`
`_HAL_RCC_UART5_CLK_DISABLE`
`_HAL_RCC_I2C1_CLK_DISABLE`
`_HAL_RCC_I2C2_CLK_DISABLE`
`_HAL_RCC_I2C3_CLK_DISABLE`
`_HAL_RCC_CEC_CLK_DISABLE`
`_HAL_RCC_DAC12_CLK_DISABLE`
`_HAL_RCC_UART7_CLK_DISABLE`
`_HAL_RCC_UART8_CLK_DISABLE`
`_HAL_RCC_CRS_CLK_DISABLE`
`_HAL_RCC_SWPMI1_CLK_DISABLE`
`_HAL_RCC_OPAMP_CLK_DISABLE`
`_HAL_RCC_MDIOS_CLK_DISABLE`
`_HAL_RCC_FDCAN_CLK_DISABLE`
`_HAL_RCC_TIM1_CLK_ENABLE`

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

`_HAL_RCC_TIM8_CLK_ENABLE`
`_HAL_RCC_USART1_CLK_ENABLE`
`_HAL_RCC_USART6_CLK_ENABLE`
`_HAL_RCC_SPI1_CLK_ENABLE`
`_HAL_RCC_SPI4_CLK_ENABLE`
`_HAL_RCC_TIM15_CLK_ENABLE`
`_HAL_RCC_TIM16_CLK_ENABLE`
`_HAL_RCC_TIM17_CLK_ENABLE`
`_HAL_RCC_SPI5_CLK_ENABLE`

```
_HAL_RCC_SAI1_CLK_ENABLE
__HAL_RCC_SAI2_CLK_ENABLE
__HAL_RCC_SAI3_CLK_ENABLE
__HAL_RCC_DFSDM1_CLK_ENABLE
__HAL_RCC_HRTIM1_CLK_ENABLE
__HAL_RCC_TIM1_CLK_DISABLE
__HAL_RCC_TIM8_CLK_DISABLE
__HAL_RCC_USART1_CLK_DISABLE
__HAL_RCC_USART6_CLK_DISABLE
__HAL_RCC_SPI1_CLK_DISABLE
__HAL_RCC_SPI4_CLK_DISABLE
__HAL_RCC_TIM15_CLK_DISABLE
__HAL_RCC_TIM16_CLK_DISABLE
__HAL_RCC_TIM17_CLK_DISABLE
__HAL_RCC_SPI5_CLK_DISABLE
__HAL_RCC_SAI1_CLK_DISABLE
__HAL_RCC_SAI2_CLK_DISABLE
__HAL_RCC_SAI3_CLK_DISABLE
__HAL_RCC_DFSDM1_CLK_DISABLE
__HAL_RCC_HRTIM1_CLK_DISABLE
__HAL_RCC_SYSCFG_CLK_ENABLE
```

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

```
_HAL_RCC_LPUART1_CLK_ENABLE
__HAL_RCC_SPI6_CLK_ENABLE
__HAL_RCC_I2C4_CLK_ENABLE
__HAL_RCC_LPTIM2_CLK_ENABLE
__HAL_RCC_LPTIM3_CLK_ENABLE
```

```
_HAL_RCC_LPTIM4_CLK_ENABLE
__HAL_RCC_LPTIM5_CLK_ENABLE
__HAL_RCC_COMP12_CLK_ENABLE
__HAL_RCC_VREF_CLK_ENABLE
__HAL_RCC_SAI4_CLK_ENABLE
__HAL_RCC_RTC_CLK_ENABLE
__HAL_RCC_SYSCFG_CLK_DISABLE
__HAL_RCC_LPUART1_CLK_DISABLE
__HAL_RCC_SPI6_CLK_DISABLE
__HAL_RCC_I2C4_CLK_DISABLE
__HAL_RCC_LPTIM2_CLK_DISABLE
__HAL_RCC_LPTIM3_CLK_DISABLE
__HAL_RCC_LPTIM4_CLK_DISABLE
__HAL_RCC_LPTIM5_CLK_DISABLE
__HAL_RCC_COMP12_CLK_DISABLE
__HAL_RCC_VREF_CLK_DISABLE
__HAL_RCC_RTC_CLK_DISABLE
__HAL_RCC_SAI4_CLK_DISABLE
__HAL_RCC_AHB3_FORCE_RESET
__HAL_RCC_MDMA_FORCE_RESET
__HAL_RCC_DMA2D_FORCE_RESET
__HAL_RCC_JPGDECRST_FORCE_RESET
__HAL_RCC_FMC_FORCE_RESET
__HAL_RCC_QSPI_FORCE_RESET
__HAL_RCC_SDMMC1_FORCE_RESET
__HAL_RCC_CPU_FORCE_RESET
__HAL_RCC_AHB3_RELEASE_RESET
__HAL_RCC_MDMA_RELEASE_RESET
```

_HAL_RCC_DMA2D_RELEASE_RESET

_HAL_RCC_JPGDECRST_RELEASE_RESET

_HAL_RCC_FMC_RELEASE_RESET

_HAL_RCC_QSPI_RELEASE_RESET

_HAL_RCC_SDMMC1_RELEASE_RESET

_HAL_RCC_CPU_RELEASE_RESET

_HAL_RCC_AHB1_FORCE_RESET

_HAL_RCC_DMA1_FORCE_RESET

_HAL_RCC_DMA2_FORCE_RESET

_HAL_RCC_ADC12_FORCE_RESET

_HAL_RCC_ETH1MAC_FORCE_RESET

_HAL_RCC_USB1_OTG_HS_FORCE_RESET

_HAL_RCC_USB2_OTG_FS_FORCE_RESET

_HAL_RCC_AHB1_RELEASE_RESET

_HAL_RCC_DMA1_RELEASE_RESET

_HAL_RCC_DMA2_RELEASE_RESET

_HAL_RCC_ADC12_RELEASE_RESET

_HAL_RCC_ETH1MAC_RELEASE_RESET

_HAL_RCC_USB1_OTG_HS_RELEASE_RESET

_HAL_RCC_USB2_OTG_FS_RELEASE_RESET

_HAL_RCC_AHB2_FORCE_RESET

_HAL_RCC_DCMI_FORCE_RESET

_HAL_RCC_CRYP_FORCE_RESET

_HAL_RCC_HASH_FORCE_RESET

_HAL_RCC_RNG_FORCE_RESET

_HAL_RCC_SDMMC2_FORCE_RESET

_HAL_RCC_AHB2_RELEASE_RESET

_HAL_RCC_DCMI_RELEASE_RESET

_HAL_RCC_CRYP_RELEASE_RESET

_HAL_RCC_HASH_RELEASE_RESET

_HAL_RCC RNG RELEASE RESET

_HAL_RCC_SDMMC2_RELEASE_RESET

_HAL_RCC_AHB4_FORCE_RESET

_HAL_RCC_GPIOA_FORCE_RESET

_HAL_RCC_GPIOB_FORCE_RESET

_HAL_RCC_GPIOC_FORCE_RESET

_HAL_RCC_GPIOD_FORCE_RESET

_HAL_RCC_GPIOE_FORCE_RESET

_HAL_RCC_GPIOF_FORCE_RESET

_HAL_RCC_GPIOG_FORCE_RESET

_HAL_RCC_GPIOH_FORCE_RESET

_HAL_RCC_GPIOI_FORCE_RESET

_HAL_RCC_GPIOJ_FORCE_RESET

_HAL_RCC_GPIOK_FORCE_RESET

_HAL_RCC_CRC_FORCE_RESET

_HAL_RCC_BDMA_FORCE_RESET

_HAL_RCC_ADC3_FORCE_RESET

_HAL_RCC_HSEM_FORCE_RESET

_HAL_RCC_AHB4_RELEASE_RESET

_HAL_RCC_GPIOA_RELEASE_RESET

_HAL_RCC_GPIOB_RELEASE_RESET

_HAL_RCC_GPIOC_RELEASE_RESET

_HAL_RCC_GPIOD_RELEASE_RESET

_HAL_RCC_GPIOE_RELEASE_RESET

_HAL_RCC_GPIOF_RELEASE_RESET

_HAL_RCC_GPIOG_RELEASE_RESET

_HAL_RCC_GPIOH_RELEASE_RESET

_HAL_RCC_GPIOI_RELEASE_RESET

_HAL_RCC_GPIOJ_RELEASE_RESET

_HAL_RCC_GPIOK_RELEASE_RESET

_HAL_RCC_CRC_RELEASE_RESET

_HAL_RCC_BDMA_RELEASE_RESET

_HAL_RCC_ADC3_RELEASE_RESET

_HAL_RCC_HSEM_RELEASE_RESET

_HAL_RCC_APB3_FORCE_RESET

_HAL_RCC_LTDC_FORCE_RESET

_HAL_RCC_APB3_RELEASE_RESET

_HAL_RCC_LTDC_RELEASE_RESET

_HAL_RCC_APB1L_FORCE_RESET

_HAL_RCC_APB1H_FORCE_RESET

_HAL_RCC_TIM2_FORCE_RESET

_HAL_RCC_TIM3_FORCE_RESET

_HAL_RCC_TIM4_FORCE_RESET

_HAL_RCC_TIM5_FORCE_RESET

_HAL_RCC_TIM6_FORCE_RESET

_HAL_RCC_TIM7_FORCE_RESET

_HAL_RCC_TIM12_FORCE_RESET

_HAL_RCC_TIM13_FORCE_RESET

_HAL_RCC_TIM14_FORCE_RESET

_HAL_RCC_LPTIM1_FORCE_RESET

_HAL_RCC_SPI2_FORCE_RESET

_HAL_RCC_SPI3_FORCE_RESET

_HAL_RCC_SPDIFRX_FORCE_RESET

_HAL_RCC_USART2_FORCE_RESET

_HAL_RCC_USART3_FORCE_RESET

_HAL_RCC_UART4_FORCE_RESET

_HAL_RCC_UART5_FORCE_RESET

_HAL_RCC_I2C1_FORCE_RESET

_HAL_RCC_I2C2_FORCE_RESET

_HAL_RCC_I2C3_FORCE_RESET

_HAL_RCC_CEC_FORCE_RESET

_HAL_RCC_DAC12_FORCE_RESET

_HAL_RCC_UART7_FORCE_RESET

_HAL_RCC_UART8_FORCE_RESET

_HAL_RCC_CRS_FORCE_RESET

_HAL_RCC_SWPMI1_FORCE_RESET

_HAL_RCC_OPAMP_FORCE_RESET

_HAL_RCC_MDIOS_FORCE_RESET

_HAL_RCC_FDCAN_FORCE_RESET

_HAL_RCC_APB1L_RELEASE_RESET

_HAL_RCC_APB1H_RELEASE_RESET

_HAL_RCC_TIM2_RELEASE_RESET

_HAL_RCC_TIM3_RELEASE_RESET

_HAL_RCC_TIM4_RELEASE_RESET

_HAL_RCC_TIM5_RELEASE_RESET

_HAL_RCC_TIM6_RELEASE_RESET

_HAL_RCC_TIM7_RELEASE_RESET

_HAL_RCC_TIM12_RELEASE_RESET

_HAL_RCC_TIM13_RELEASE_RESET

_HAL_RCC_TIM14_RELEASE_RESET

_HAL_RCC_LPTIM1_RELEASE_RESET

_HAL_RCC_SPI2_RELEASE_RESET

_HAL_RCC_SPI3_RELEASE_RESET

_HAL_RCC_SPDIFRX_RELEASE_RESET

_HAL_RCC_USART2_RELEASE_RESET

_HAL_RCC_USART3_RELEASE_RESET

_HAL_RCC_UART4_RELEASE_RESET

_HAL_RCC_UART5_RELEASE_RESET

_HAL_RCC_I2C1_RELEASE_RESET

_HAL_RCC_I2C2_RELEASE_RESET

_HAL_RCC_I2C3_RELEASE_RESET

_HAL_RCC_CEC_RELEASE_RESET

_HAL_RCC_DAC12_RELEASE_RESET

_HAL_RCC_UART7_RELEASE_RESET

_HAL_RCC_UART8_RELEASE_RESET

_HAL_RCC_CRS_RELEASE_RESET

_HAL_RCC_SWPMI1_RELEASE_RESET

_HAL_RCC_OPAMP_RELEASE_RESET

_HAL_RCC_MDIOS_RELEASE_RESET

_HAL_RCC_FDCAN_RELEASE_RESET

_HAL_RCC_APB2_FORCE_RESET

_HAL_RCC_TIM1_FORCE_RESET

_HAL_RCC_TIM8_FORCE_RESET

_HAL_RCC_USART1_FORCE_RESET

_HAL_RCC_USART6_FORCE_RESET

_HAL_RCC_SPI1_FORCE_RESET

_HAL_RCC_SPI4_FORCE_RESET

_HAL_RCC_TIM15_FORCE_RESET

_HAL_RCC_TIM16_FORCE_RESET

_HAL_RCC_TIM17_FORCE_RESET

_HAL_RCC_SPI5_FORCE_RESET

_HAL_RCC_SAI1_FORCE_RESET

_HAL_RCC_SAI2_FORCE_RESET

_HAL_RCC_SAI3_FORCE_RESET

_HAL_RCC_DFSDM1_FORCE_RESET

_HAL_RCC_HRTIM1_FORCE_RESET

_HAL_RCC_APB2_RELEASE_RESET

_HAL_RCC_TIM1_RELEASE_RESET

_HAL_RCC_TIM8_RELEASE_RESET

_HAL_RCC_USART1_RELEASE_RESET

_HAL_RCC_USART6_RELEASE_RESET

_HAL_RCC_SPI1_RELEASE_RESET

_HAL_RCC_SPI4_RELEASE_RESET

_HAL_RCC_TIM15_RELEASE_RESET

_HAL_RCC_TIM16_RELEASE_RESET

_HAL_RCC_TIM17_RELEASE_RESET

_HAL_RCC_SPI5_RELEASE_RESET

_HAL_RCC_SAI1_RELEASE_RESET

_HAL_RCC_SAI2_RELEASE_RESET

_HAL_RCC_SAI3_RELEASE_RESET

_HAL_RCC_DFSDM1_RELEASE_RESET

_HAL_RCC_HRTIM1_RELEASE_RESET

_HAL_RCC_APB4_FORCE_RESET

_HAL_RCC_SYSCFG_FORCE_RESET

_HAL_RCC_LPUART1_FORCE_RESET

_HAL_RCC_SPI6_FORCE_RESET

_HAL_RCC_I2C4_FORCE_RESET

_HAL_RCC_LPTIM2_FORCE_RESET

```
_HAL_RCC_LPTIM3_FORCE_RESET
__HAL_RCC_LPTIM4_FORCE_RESET
__HAL_RCC_LPTIM5_FORCE_RESET
__HAL_RCC_COMP12_FORCE_RESET
__HAL_RCC_VREF_FORCE_RESET
__HAL_RCC_SAI4_FORCE_RESET
__HAL_RCC_APB4_RELEASE_RESET
__HAL_RCC_SYSCFG_RELEASE_RESET
__HAL_RCC_LPUART1_RELEASE_RESET
__HAL_RCC_SPI6_RELEASE_RESET
__HAL_RCC_I2C4_RELEASE_RESET
__HAL_RCC_LPTIM2_RELEASE_RESET
__HAL_RCC_LPTIM3_RELEASE_RESET
__HAL_RCC_LPTIM4_RELEASE_RESET
__HAL_RCC_LPTIM5_RELEASE_RESET
__HAL_RCC_COMP12_RELEASE_RESET
__HAL_RCC_VREF_RELEASE_RESET
__HAL_RCC_SAI4_RELEASE_RESET
__HAL_RCC_MDMA_CLK_SLEEP_ENABLE
```

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

```
_HAL_RCC_DMA2D_CLK_SLEEP_ENABLE
__HAL_RCC_JPGDEC_CLK_SLEEP_ENABLE
__HAL_RCC_FLASH_CLK_SLEEP_ENABLE
__HAL_RCC_FMC_CLK_SLEEP_ENABLE
__HAL_RCC_QSPI_CLK_SLEEP_ENABLE
__HAL_RCC_SDMMC1_CLK_SLEEP_ENABLE
```

```
_HAL_RCC_DTCM1_CLK_SLEEP_ENABLE
_HAL_RCC_DTCM2_CLK_SLEEP_ENABLE
_HAL_RCC_ITCM_CLK_SLEEP_ENABLE
_HAL_RCC_D1SRAM1_CLK_SLEEP_ENABLE
_HAL_RCC_MDMA_CLK_SLEEP_DISABLE
_HAL_RCC_DMA2D_CLK_SLEEP_DISABLE
_HAL_RCC_JPGDEC_CLK_SLEEP_DISABLE
_HAL_RCC_FLASH_CLK_SLEEP_DISABLE
_HAL_RCC_FMC_CLK_SLEEP_DISABLE
_HAL_RCC_QSPI_CLK_SLEEP_DISABLE
_HAL_RCC_SDMMC1_CLK_SLEEP_DISABLE
_HAL_RCC_DTCM1_CLK_SLEEP_DISABLE
_HAL_RCC_DTCM2_CLK_SLEEP_DISABLE
_HAL_RCC_ITCM_CLK_SLEEP_DISABLE
_HAL_RCC_D1SRAM1_CLK_SLEEP_DISABLE
_HAL_RCC_DMA1_CLK_SLEEP_ENABLE
```

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

```
_HAL_RCC_DMA2_CLK_SLEEP_ENABLE
_HAL_RCC_ADC12_CLK_SLEEP_ENABLE
_HAL_RCC_ETH1MAC_CLK_SLEEP_ENABLE
_HAL_RCC_ETH1TX_CLK_SLEEP_ENABLE
_HAL_RCC_ETH1RX_CLK_SLEEP_ENABLE
_HAL_RCC_ETH1PTP_CLK_SLEEP_ENABLE
_HAL_RCC_USB1_OTG_HS_CLK_SLEEP_ENABLE
_HAL_RCC_USB1_OTG_HS_ULPI_CLK_SLEEP_ENABLE
_HAL_RCC_USB2_OTG_FS_CLK_SLEEP_ENABLE
```

```
_HAL_RCC_USB2_OTG_FS_ULPI_CLK_SLEEP_ENABLE  
  
_HAL_RCC_DMA1_CLK_SLEEP_DISABLE  
  
_HAL_RCC_DMA2_CLK_SLEEP_DISABLE  
  
_HAL_RCC_ADC12_CLK_SLEEP_DISABLE  
  
_HAL_RCC_ETH1MAC_CLK_SLEEP_DISABLE  
  
_HAL_RCC_ETH1TX_CLK_SLEEP_DISABLE  
  
_HAL_RCC_ETH1RX_CLK_SLEEP_DISABLE  
  
_HAL_RCC_ETH1PTP_CLK_SLEEP_DISABLE  
  
_HAL_RCC_USB1_OTG_HS_CLK_SLEEP_DISABLE  
  
_HAL_RCC_USB1_OTG_HS_ULPI_CLK_SLEEP_DISABLE  
  
_HAL_RCC_USB2_OTG_FS_CLK_SLEEP_DISABLE  
  
_HAL_RCC_USB2_OTG_FS_ULPI_CLK_SLEEP_DISABLE  
  
_HAL_RCC_DCMI_CLK_SLEEP_ENABLE
```

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

```
_HAL_RCC_CRYP_CLK_SLEEP_ENABLE  
  
_HAL_RCC_HASH_CLK_SLEEP_ENABLE  
  
_HAL_RCC RNG_CLK_SLEEP_ENABLE  
  
_HAL_RCC_SDMMC2_CLK_SLEEP_ENABLE  
  
_HAL_RCC_D2SRAM1_CLK_SLEEP_ENABLE  
  
_HAL_RCC_D2SRAM2_CLK_SLEEP_ENABLE  
  
_HAL_RCC_D2SRAM3_CLK_SLEEP_ENABLE  
  
_HAL_RCC_DCMI_CLK_SLEEP_DISABLE  
  
_HAL_RCC_CRYP_CLK_SLEEP_DISABLE  
  
_HAL_RCC_HASH_CLK_SLEEP_DISABLE  
  
_HAL_RCC RNG_CLK_SLEEP_DISABLE  
  
_HAL_RCC_SDMMC2_CLK_SLEEP_DISABLE
```

`_HAL_RCC_D2SRAM1_CLK_SLEEP_DISABLE`
`_HAL_RCC_D2SRAM2_CLK_SLEEP_DISABLE`
`_HAL_RCC_D2SRAM3_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOA_CLK_SLEEP_ENABLE`

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

`_HAL_RCC_GPIOB_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOC_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOD_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOE_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOF_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOG_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOH_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOI_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOJ_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOK_CLK_SLEEP_ENABLE`
`_HAL_RCC_CRC_CLK_SLEEP_ENABLE`
`_HAL_RCC_BDMA_CLK_SLEEP_ENABLE`
`_HAL_RCC_ADC3_CLK_SLEEP_ENABLE`
`_HAL_RCC_BKPRAM_CLK_SLEEP_ENABLE`
`_HAL_RCC_D3SRAM1_CLK_SLEEP_ENABLE`
`_HAL_RCC_GPIOA_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOB_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOC_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOD_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOE_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOF_CLK_SLEEP_DISABLE`

`_HAL_RCC_GPIOG_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOH_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOI_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOJ_CLK_SLEEP_DISABLE`
`_HAL_RCC_GPIOK_CLK_SLEEP_DISABLE`
`_HAL_RCC_CRC_CLK_SLEEP_DISABLE`
`_HAL_RCC_BDMA_CLK_SLEEP_DISABLE`
`_HAL_RCC_ADC3_CLK_SLEEP_DISABLE`
`_HAL_RCC_BKPRAM_CLK_SLEEP_DISABLE`
`_HAL_RCC_D3SRAM1_CLK_SLEEP_DISABLE`
`_HAL_RCC_LTDC_CLK_SLEEP_ENABLE`

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

`_HAL_RCC_WWDG1_CLK_SLEEP_ENABLE`
`_HAL_RCC_LTDC_CLK_SLEEP_DISABLE`
`_HAL_RCC_WWDG1_CLK_SLEEP_DISABLE`
`_HAL_RCC_TIM2_CLK_SLEEP_ENABLE`

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

`_HAL_RCC_TIM3_CLK_SLEEP_ENABLE`
`_HAL_RCC_TIM4_CLK_SLEEP_ENABLE`
`_HAL_RCC_TIM5_CLK_SLEEP_ENABLE`
`_HAL_RCC_TIM6_CLK_SLEEP_ENABLE`
`_HAL_RCC_TIM7_CLK_SLEEP_ENABLE`
`_HAL_RCC_TIM12_CLK_SLEEP_ENABLE`
`_HAL_RCC_TIM13_CLK_SLEEP_ENABLE`
`_HAL_RCC_TIM14_CLK_SLEEP_ENABLE`

```
_HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE
__HAL_RCC_SPI2_CLK_SLEEP_ENABLE
__HAL_RCC_SPI3_CLK_SLEEP_ENABLE
__HAL_RCC_SPDIFRX_CLK_SLEEP_ENABLE
__HAL_RCC_USART2_CLK_SLEEP_ENABLE
__HAL_RCC_USART3_CLK_SLEEP_ENABLE
__HAL_RCC_UART4_CLK_SLEEP_ENABLE
__HAL_RCC_UART5_CLK_SLEEP_ENABLE
__HAL_RCC_I2C1_CLK_SLEEP_ENABLE
__HAL_RCC_I2C2_CLK_SLEEP_ENABLE
__HAL_RCC_I2C3_CLK_SLEEP_ENABLE
__HAL_RCC_CEC_CLK_SLEEP_ENABLE
__HAL_RCC_DAC12_CLK_SLEEP_ENABLE
__HAL_RCC_UART7_CLK_SLEEP_ENABLE
__HAL_RCC_UART8_CLK_SLEEP_ENABLE
__HAL_RCC_CRS_CLK_SLEEP_ENABLE
__HAL_RCC_SWPMI1_CLK_SLEEP_ENABLE
__HAL_RCC_OPAMP_CLK_SLEEP_ENABLE
__HAL_RCC_MDIOS_CLK_SLEEP_ENABLE
__HAL_RCC_FDCAN_CLK_SLEEP_ENABLE
__HAL_RCC_TIM2_CLK_SLEEP_DISABLE
__HAL_RCC_TIM3_CLK_SLEEP_DISABLE
__HAL_RCC_TIM4_CLK_SLEEP_DISABLE
__HAL_RCC_TIM5_CLK_SLEEP_DISABLE
__HAL_RCC_TIM6_CLK_SLEEP_DISABLE
__HAL_RCC_TIM7_CLK_SLEEP_DISABLE
__HAL_RCC_TIM12_CLK_SLEEP_DISABLE
__HAL_RCC_TIM13_CLK_SLEEP_DISABLE
```

```
_HAL_RCC_TIM14_CLK_SLEEP_DISABLE
_HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE
_HAL_RCC_SPI2_CLK_SLEEP_DISABLE
_HAL_RCC_SPI3_CLK_SLEEP_DISABLE
_HAL_RCC_SPDIFRX_CLK_SLEEP_DISABLE
_HAL_RCC_USART2_CLK_SLEEP_DISABLE
_HAL_RCC_USART3_CLK_SLEEP_DISABLE
_HAL_RCC_UART4_CLK_SLEEP_DISABLE
_HAL_RCC_UART5_CLK_SLEEP_DISABLE
_HAL_RCC_I2C1_CLK_SLEEP_DISABLE
_HAL_RCC_I2C2_CLK_SLEEP_DISABLE
_HAL_RCC_I2C3_CLK_SLEEP_DISABLE
_HAL_RCC_CEC_CLK_SLEEP_DISABLE
_HAL_RCC_DAC12_CLK_SLEEP_DISABLE
_HAL_RCC_UART7_CLK_SLEEP_DISABLE
_HAL_RCC_UART8_CLK_SLEEP_DISABLE
_HAL_RCC_CRS_CLK_SLEEP_DISABLE
_HAL_RCC_SWPMI1_CLK_SLEEP_DISABLE
_HAL_RCC_OPAMP_CLK_SLEEP_DISABLE
_HAL_RCC_MDIOS_CLK_SLEEP_DISABLE
_HAL_RCC_FDCAN_CLK_SLEEP_DISABLE
_HAL_RCC_TIM1_CLK_SLEEP_ENABLE
```

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

```
_HAL_RCC_TIM8_CLK_SLEEP_ENABLE
_HAL_RCC_USART1_CLK_SLEEP_ENABLE
_HAL_RCC_USART6_CLK_SLEEP_ENABLE
```

```
_HAL_RCC_SPI1_CLK_SLEEP_ENABLE
__HAL_RCC_SPI4_CLK_SLEEP_ENABLE
__HAL_RCC_TIM15_CLK_SLEEP_ENABLE
__HAL_RCC_TIM16_CLK_SLEEP_ENABLE
__HAL_RCC_TIM17_CLK_SLEEP_ENABLE
__HAL_RCC_SPI5_CLK_SLEEP_ENABLE
__HAL_RCC_SAI1_CLK_SLEEP_ENABLE
__HAL_RCC_SAI2_CLK_SLEEP_ENABLE
__HAL_RCC_SAI3_CLK_SLEEP_ENABLE
__HAL_RCC_DFSDM1_CLK_SLEEP_ENABLE
__HAL_RCC_HRTIM1_CLK_SLEEP_ENABLE
__HAL_RCC_TIM1_CLK_SLEEP_DISABLE
__HAL_RCC_TIM8_CLK_SLEEP_DISABLE
__HAL_RCC_USART1_CLK_SLEEP_DISABLE
__HAL_RCC_USART6_CLK_SLEEP_DISABLE
__HAL_RCC_SPI1_CLK_SLEEP_DISABLE
__HAL_RCC_SPI4_CLK_SLEEP_DISABLE
__HAL_RCC_TIM15_CLK_SLEEP_DISABLE
__HAL_RCC_TIM16_CLK_SLEEP_DISABLE
__HAL_RCC_TIM17_CLK_SLEEP_DISABLE
__HAL_RCC_SPI5_CLK_SLEEP_DISABLE
__HAL_RCC_SAI1_CLK_SLEEP_DISABLE
__HAL_RCC_SAI2_CLK_SLEEP_DISABLE
__HAL_RCC_SAI3_CLK_SLEEP_DISABLE
__HAL_RCC_DFSDM1_CLK_SLEEP_DISABLE
__HAL_RCC_HRTIM1_CLK_SLEEP_DISABLE
__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE
```

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is ENABLEd again. By default, all peripheral clocks are ENABLEd during SLEEP mode.

`_HAL_RCC_LPUART1_CLK_SLEEP_ENABLE`
`_HAL_RCC_SPI6_CLK_SLEEP_ENABLE`
`_HAL_RCC_I2C4_CLK_SLEEP_ENABLE`
`_HAL_RCC_LPTIM2_CLK_SLEEP_ENABLE`
`_HAL_RCC_LPTIM3_CLK_SLEEP_ENABLE`
`_HAL_RCC_LPTIM4_CLK_SLEEP_ENABLE`
`_HAL_RCC_LPTIM5_CLK_SLEEP_ENABLE`
`_HAL_RCC_COMP12_CLK_SLEEP_ENABLE`
`_HAL_RCC_VREF_CLK_SLEEP_ENABLE`
`_HAL_RCC_RTC_CLK_SLEEP_ENABLE`
`_HAL_RCC_SAI4_CLK_SLEEP_ENABLE`
`_HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE`
`_HAL_RCC_LPUART1_CLK_SLEEP_DISABLE`
`_HAL_RCC_SPI6_CLK_SLEEP_DISABLE`
`_HAL_RCC_I2C4_CLK_SLEEP_DISABLE`
`_HAL_RCC_LPTIM2_CLK_SLEEP_DISABLE`
`_HAL_RCC_LPTIM3_CLK_SLEEP_DISABLE`
`_HAL_RCC_LPTIM4_CLK_SLEEP_DISABLE`
`_HAL_RCC_LPTIM5_CLK_SLEEP_DISABLE`
`_HAL_RCC_COMP12_CLK_SLEEP_DISABLE`
`_HAL_RCC_VREF_CLK_SLEEP_DISABLE`
`_HAL_RCC_RTC_CLK_SLEEP_DISABLE`
`_HAL_RCC_SAI4_CLK_SLEEP_DISABLE`
`_HAL_RCC_BDMA_CLKAM_ENABLE`

Notes:

- After reset (default config), peripheral clock is disabled when CPU is in CSTOP

_HAL_RCC_LPUART1_CLKAM_ENABLE

_HAL_RCC_SPI6_CLKAM_ENABLE

_HAL_RCC_I2C4_CLKAM_ENABLE

_HAL_RCC_LPTIM2_CLKAM_ENABLE

_HAL_RCC_LPTIM3_CLKAM_ENABLE

_HAL_RCC_LPTIM4_CLKAM_ENABLE

_HAL_RCC_LPTIM5_CLKAM_ENABLE

_HAL_RCC_COMP12_CLKAM_ENABLE

_HAL_RCC_VREF_CLKAM_ENABLE

_HAL_RCC_RTC_CLKAM_ENABLE

_HAL_RCC_CRC_CLKAM_ENABLE

_HAL_RCC_SAI4_CLKAM_ENABLE

_HAL_RCC_ADC3_CLKAM_ENABLE

_HAL_RCC_BKPRAM_CLKAM_ENABLE

_HAL_RCC_D3SRAM1_CLKAM_ENABLE

_HAL_RCC_BDMA_CLKAM_DISABLE

_HAL_RCC_LPUART1_CLKAM_DISABLE

_HAL_RCC_SPI6_CLKAM_DISABLE

_HAL_RCC_I2C4_CLKAM_DISABLE

_HAL_RCC_LPTIM2_CLKAM_DISABLE

_HAL_RCC_LPTIM3_CLKAM_DISABLE

_HAL_RCC_LPTIM4_CLKAM_DISABLE

_HAL_RCC_LPTIM5_CLKAM_DISABLE

_HAL_RCC_COMP12_CLKAM_DISABLE

_HAL_RCC_VREF_CLKAM_DISABLE

_HAL_RCC_RTC_CLKAM_DISABLE

_HAL_RCC_CRC_CLKAM_DISABLE

_HAL_RCC_SAI4_CLKAM_DISABLE

__HAL_RCC_ADC3_CLKAM_DISABLE

__HAL_RCC_BKPRAM_CLKAM_DISABLE

__HAL_RCC_D3SRAM1_CLKAM_DISABLE

__HAL_RCC_HSI_CONFIG

Description:

- Macro to enable or disable the Internal High Speed oscillator (HSI).

Parameters:

- __STATE__: specifies the new state of the HSI. This parameter can be one of the following values:
 - RCC_HSI_OFF turn OFF the HSI oscillator
 - RCC_HSI_ON turn ON the HSI oscillator
 - RCC_HSI_DIV1 turn ON the HSI oscillator and divide it by 1 (default after reset)
 - RCC_HSI_DIV2 turn ON the HSI oscillator and divide it by 2
 - RCC_HSI_DIV4 turn ON the HSI oscillator and divide it by 4
 - RCC_HSI_DIV8 turn ON the HSI oscillator and divide it by 8

Notes:

- After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used to clock the PLL and/or system clock. HSI can not be stopped if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then stop the HSI. The HSI is stopped by hardware when entering STOP and STANDBY modes.
- When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_GET_HSI_DIVIDER

Description:

- Macro to get the HSI divider.

Return value:

- The: HSI divider. The returned value can be one of the following:
 - RCC_CR_HSIDIV_1 HSI oscillator divided by 1 (default after reset)
 - RCC_CR_HSIDIV_2 HSI oscillator divided by 2
 - RCC_CR_HSIDIV_4 HSI oscillator divided by 4
 - RCC_CR_HSIDIV_8 HSI oscillator divided by 8

__HAL_RCC_HSI_ENABLE

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after start-up from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This parameter can be: ENABLE or DISABLE. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal High Speed oscillator (HSI) calibration value.

Parameters:

- HSICalibrationValue: specifies the calibration trimming value. This parameter must be a number between 0 and 0x3F.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

[__HAL_RCC_HSISTOP_ENABLE](#)**Description:**

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for some peripherals.

Return value:

- None

Notes:

- Keeping the HSI ON in STOP mode allows to avoid slowing down the communication speed because of the HSI start-up time. The enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE.

[__HAL_RCC_HSISTOP_DISABLE](#)[__HAL_RCC_HSI48_ENABLE](#)**Notes:**

- After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable and can be used to clock the USB. The HSI48 is stopped by hardware when entering STOP and STANDBY modes.

[__HAL_RCC_HSI48_DISABLE](#)[__HAL_RCC_CSI_ENABLE](#)**Notes:**

- The CSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after start-up from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). CSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the CSI. After enabling the CSI, the application software should wait on CSIRDY flag to be set indicating that CSI clock is stable and can be used as system clock source. When the CSI is stopped, CSIRDY flag goes low after 6 CSI oscillator clock cycles.

[__HAL_RCC_CSI_DISABLE](#)[__HAL_RCC_CSI_CALIBRATIONVALUE_ADJUST](#)**Description:**

- Macro Adjusts the Internal oscillator (CSI) calibration value.

Parameters:

- CSICalibrationValue: specifies the calibration trimming value. This parameter must be a number between 0 and 0x1F.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal CSI RC.

__HAL_RCC_CSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Low-power Internal oscillator (CSI) in STOP mode to be quickly available as kernel clock for USARTs and I2Cs.

Return value:

- None

Notes:

- Keeping the CSI ON in STOP mode allows to avoid slowing down the communication speed because of the CSI start-up time. The enable of this function has not effect on the CSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_CSISTOP_DISABLE

__HAL_RCC_LSI_ENABLE

Notes:

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

__HAL_RCC_HSE_CONFIG

Description:

- Macro to configure the External High Speed oscillator (__HSE__).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
 - RCC_HSE_OFF: turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
 - RCC_HSE_ON: turn ON the HSE oscillator.
 - RCC_HSE_BYPASS: HSE oscillator bypassed with external clock.

Notes:

- After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the clock security system(CSS) was previously enabled you have to enable it again after calling this function.

__HAL_RCC_RTC_ENABLE

Notes:

- These macros must be used only after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

__HAL_RCC_RTC_CLKPRESCALER

Description:

- Macros to configure the RTC clock (RTCCLK).

Parameters:

- `__RTCCLKSource__`: specifies the RTC clock source. This parameter can be one of the following values:
 - `RCC_RTCCLKSOURCE_LSE`: LSE selected as RTC clock.
 - `RCC_RTCCLKSOURCE_LSI`: LSI selected as RTC clock.
 - `RCC_RTCCLKSOURCE_HSE_DIVx`: HSE clock divided by x selected as RTC clock, where x:[2,31]

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it can't be changed unless the Backup domain is reset using `__HAL_RCC_BackupReset_RELEASE()` macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

[`__HAL_RCC_RTC_CONFIG`](#)[`__HAL_RCC_GET_RTC_SOURCE`](#)[`__HAL_RCC_BACKUPRESET_FORCE`](#)**Notes:**

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_CSR register. The BKPSRAM is not affected by this reset.

[`__HAL_RCC_BACKUPRESET_RELEASE`](#)[`__HAL_RCC_PLL_ENABLE`](#)**Notes:**

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source. The main PLL is disabled by hardware when entering STOP and STANDBY modes.

[`__HAL_RCC_PLL_DISABLE`](#)[`__HAL_RCC_PLLCLKOUT_ENABLE`](#)**Description:**

- Enables or disables each clock output (PLL_P_CLK, PLL_Q_CLK, PLL_R_CLK)

Parameters:

- `__RCC_PLL1ClockOut__`: specifies the PLL clock to be outputted. This parameter can be one of the following values:
 - `RCC_PLL1_DIVP`: This clock is used to generate system clock (up to 400MHZ)
 - `RCC_PLL1_DIVQ`: This clock is used to generate peripherals clock (up to 400MHZ)
 - `RCC_PLL1_DIVR`: This clock is used to generate peripherals clock (up to 400MHZ)

Return value:

- None

Notes:

- Enabling/disabling Those Clocks can be any time without the need to stop the PLL, (except the ck_pll_p of the System PLL that cannot be stopped if used as System Clock. This is mainly used to save Power).

[`__HAL_RCC_PLLCLKOUT_DISABLE`](#)

__HAL_RCC_PLLFRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL1 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be any time without the need to stop the PLL1

__HAL_RCC_PLLFRACN_DISABLE

__HAL_RCC_PLL_CONFIG

Description:

- Macro to configures the main PLL clock source, multiplication and division factors.

Parameters:

- __RCC_PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
 - RCC_PLLSOURCE_CSI: CSI oscillator clock selected as PLL clock entry
 - RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL clock entry
 - RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL clock entry
- __PLLM1__: specifies the division factor for PLL VCO input clock This parameter must be a number between 1 and 63.
- __PLLN1__: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 4 and 512.
- __PLLP1__: specifies the division factor for system clock. This parameter must be a number between 2 and 128 (where odd numbers not allowed)
- __PLLQ1__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLLR1__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None

Notes:

- This function must be used only when the main PLL is disabled.
- This clock source (__RCC_PLLSource__) is common for the main PLL1 (main PLL) and PLL2 & PLL3 .
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 16 MHz.
- You have to set the PLLN parameter correctly to ensure that the VCO output frequency is between 150 and 420 MHz (when in medium VCO range) or between 192 and 836 MHZ (when in wide VCO range)

__HAL_RCC_PLL_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLLs clock source.

Parameters:

- __PLLSOURCE__: specifies the PLLs entry clock source. This parameter can be one of the following values:
 - RCC_PLLSOURCE_CSI: CSI oscillator clock selected as PLL clock entry
 - RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL clock entry
 - RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL clock entry

Notes:

- This function must be used only when all PLLs are disabled.

[__HAL_RCC_PLLFRACN_CONFIG](#)

Description:

- Macro to configures the main PLL clock Fractional Part Of The Multiplication Factor.

Parameters:

- RCC_PLL1FRACN: specifies Fractional Part Of The Multiplication Factor for PLL1 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL1 VCO
- Warning: The software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL1VCOSEL = 0 150 to 420 MHz if PLL1VCOSEL = 1.

[__HAL_RCC_PLL_VCIRANGE](#)

Description:

- Macro to select the PLL1 reference frequency range.

Parameters:

- RCC_PLL1VCIRANGE: specifies the PLL1 input frequency range This parameter can be one of the following values:
 - RCC_PLL1VCIRANGE_0: Range frequency is between 1 and 2 MHz
 - RCC_PLL1VCIRANGE_1: Range frequency is between 2 and 4 MHz
 - RCC_PLL1VCIRANGE_2: Range frequency is between 4 and 8 MHz
 - RCC_PLL1VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

[__HAL_RCC_PLL_VCORANGE](#)

Description:

- Macro to select the PLL1 reference frequency range.

Parameters:

- RCC_PLL1VCORANGE: specifies the PLL1 input frequency range This parameter can be one of the following values:
 - RCC_PLL1VCOWIDE: Range frequency is between 192 and 836 MHz
 - RCC_PLL1VCOMEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

[__HAL_RCC_GET_SYSCLK_SOURCE](#)

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
 - RCC_CFGR_SWS_CSI: CSI used as system clock.
 - RCC_CFGR_SWS_HSI: HSI used as system clock.

- RCC_CFGR_SWS_HSE: HSE used as system clock.
- RCC_CFGR_SWS_PLL: PLL used as system clock.

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __RCC_SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
 - RCC_SYSCLKSOURCE_HSI: HSI oscillator is used as system clock source.
 - RCC_SYSCLKSOURCE_CSI: CSI oscillator is used as system clock source.
 - RCC_SYSCLKSOURCE_HSE: HSE oscillator is used as system clock source.
 - RCC_SYSCLKSOURCE_PLLCLK: PLL output is used as system clock source.

__HAL_RCC_GET_PLL_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following:
 - RCC_PLLSOURCE_NONE: No oscillator is used as PLL clock source.
 - RCC_PLLSOURCE_CSI: CSI oscillator is used as PLL clock source.
 - RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
 - RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_LSEDRIVE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE) drive capability.

Parameters:

- __LSEDRIVE__: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
 - RCC_LSEDRIVE_LOW: LSE oscillator low drive capability.
 - RCC_LSEDRIVE_MEDIUMLOW: LSE oscillator medium low drive capability.
 - RCC_LSEDRIVE_MEDIUMHIGH: LSE oscillator medium high drive capability.
 - RCC_LSEDRIVE_HIGH: LSE oscillator high drive capability.

Return value:

- None

Notes:

- As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset).

__HAL_RCC_WAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configure the wake up from stop clock.

Parameters:

- __RCC_STOPWUCLK__: specifies the clock source used after wake up from stop This parameter can be one of the following values:
 - RCC_STOP_WAKEUPCLOCK_CSI: CSI selected as system clock source

- RCC_STOP_WAKEUPCLOCK_HSI: HSI selected as system clock source

Return value:

- None

_HAL_RCC_KERWAKEUPSTOP_CLK_CONFIG**Description:**

- Macro to configure the Kernel wake up from stop clock.

Parameters:

- _RCC_STOPKERWUCLK_: specifies the Kernel clock source used after wake up from stop This parameter can be one of the following values:
 - RCC_STOP_KERWAKEUPCLOCK_CSI: CSI selected as Kernel clock source
 - RCC_STOP_KERWAKEUPCLOCK_HSI: HSI selected as Kernel clock source

Return value:

- None

RCC_GET_PLL_OSCSOURCE

RCC Flag

RCC_FLAG_HSIRDY**RCC_FLAG_HSIDIV****RCC_FLAG_CSIRDY****RCC_FLAG_HSI48RDY****RCC_FLAG_D1CKRDY****RCC_FLAG_D2CKRDY****RCC_FLAG_HSERDY****RCC_FLAG_PLLRDY****RCC_FLAG_PLL2RDY****RCC_FLAG_PLL3RDY****RCC_FLAG_LSERDY****RCC_FLAG_LSIRDY****RCC_FLAG_RMVF****RCC_FLAG_CPURST****RCC_FLAG_D1RST****RCC_FLAG_D2RST****RCC_FLAG_BORRST**

`RCC_FLAG_PINRST`

`RCC_FLAG_PORRST`

`RCC_FLAG_SFTRST`

`RCC_FLAG_IWDG1RST`

`RCC_FLAG_WWDG1RST`

`RCC_FLAG_LPWR1RST`

`RCC_FLAG_LPWR2RST`

Flags Interrupts Management

`_HAL_RCC_ENABLE_IT`

Description:

- Enable RCC interrupt.

Parameters:

- `_INTERRUPT_`: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
 - `RCC_IT_LSIRDY`: LSI ready interrupt
 - `RCC_IT_LSERDY`: LSE ready interrupt
 - `RCC_IT_CSIRDY`: HSI ready interrupt
 - `RCC_IT_HSIRDY`: HSI ready interrupt
 - `RCC_IT_HSERDY`: HSE ready interrupt
 - `RCC_IT_HSI48RDY`: HSI48 ready interrupt
 - `RCC_IT_PLLRDY`: main PLL ready interrupt
 - `RCC_IT_PLL2RDY`: PLL2 ready interrupt
 - `RCC_IT_PLL3RDY`: PLL3 ready interrupt
 - `RCC_IT_LSECSS`: Clock security system interrupt

`_HAL_RCC_DISABLE_IT`

Description:

- Disable RCC interrupt.

Parameters:

- `_INTERRUPT_`: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
 - `RCC_IT_LSIRDY`: LSI ready interrupt
 - `RCC_IT_LSERDY`: LSE ready interrupt
 - `RCC_IT_CSIRDY`: HSI ready interrupt
 - `RCC_IT_HSIRDY`: HSI ready interrupt
 - `RCC_IT_HSERDY`: HSE ready interrupt
 - `RCC_IT_HSI48RDY`: HSI48 ready interrupt
 - `RCC_IT_PLLRDY`: main PLL ready interrupt
 - `RCC_IT_PLL2RDY`: PLL2 ready interrupt
 - `RCC_IT_PLL3RDY`: PLL3 ready interrupt
 - `RCC_IT_LSECSS`: Clock security system interrupt

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt
 - RCC_IT_LSERDY: LSE ready interrupt
 - RCC_IT_CSIRDY: CSI ready interrupt
 - RCC_IT_HSIRDY: HSI ready interrupt
 - RCC_IT_HSERDY: HSE ready interrupt
 - RCC_IT_HSI48RDY: HSI48 ready interrupt
 - RCC_IT_PLLRDY: main PLL ready interrupt
 - RCC_IT_PLL2RDY: PLL2 ready interrupt
 - RCC_IT_PLL3RDY: PLL3 ready interrupt
 - RCC_IT_HSECSS: HSE Clock Security interrupt
 - RCC_IT_LSECSS: Clock security system interrupt

__HAL_RCC_GET_IT

Description:

- Check the RCC's interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source to check. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt
 - RCC_IT_LSERDY: LSE ready interrupt
 - RCC_IT_CSIRDY: CSI ready interrupt
 - RCC_IT_HSIRDY: HSI ready interrupt
 - RCC_IT_HSERDY: HSE ready interrupt
 - RCC_IT_HSI48RDY: HSI48 ready interrupt
 - RCC_IT_PLLRDY: main PLL ready interrupt
 - RCC_IT_PLL2RDY: PLL2 ready interrupt
 - RCC_IT_PLL3RDY: PLL3 ready interrupt
 - RCC_IT_HSECSS: HSE Clock Security interrupt
 - RCC_IT_LSECSS: Clock security system interrupt

Return value:

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

__HAL_RCC_CLEAR_RESET_FLAGS

RCC_FLAG_MASK

Description:

- Check RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - RCC_FLAG_HSIRDY: HSI oscillator clock ready
 - RCC_FLAG_HSIDIV: HSI divider flag
 - RCC_FLAG_CSIRDY: CSI oscillator clock ready

- RCC_FLAG_HSI48RDY: HSI48 oscillator clock ready
- RCC_FLAG_HSERDY: HSE oscillator clock ready
- RCC_FLAG_D1CKRDY: Domain1 clock ready
- RCC_FLAG_D2CKRDY: Domain2 clock ready
- RCC_FLAG_PLLRDY: PLL1 clock ready
- RCC_FLAG_PLL2RDY: PLL2 clock ready
- RCC_FLAG_PLL3RDY: PLL3 clock ready
- RCC_FLAG_LSERDY: LSE oscillator clock ready
- RCC_FLAG_LSIRDY: LSI oscillator clock ready
- RCC_FLAG_RMVF: Remove reset Flag
- RCC_FLAG_CPURST: CPU reset flag
- RCC_FLAG_D1RST: D1 domain power switch reset flag
- RCC_FLAG_D2RST: D2 domain power switch reset flag
- RCC_FLAG_BORRST: BOR reset flag
- RCC_FLAG_PINRST: Pin reset
- RCC_FLAG_PORRST: POR/PDR reset
- RCC_FLAG_SFTRST: System reset from CPU reset flag
- RCC_FLAG_BORRST: D2 domain power switch reset flag
- RCC_FLAG_IWDG1RST: CPU Independent Watchdog reset
- RCC_FLAG_WWDG1RST: Window Watchdog1 reset
- RCC_FLAG_LPWR1RST: Reset due to illegal D1 DSTANDBY or CPU CSTOP flag
- RCC_FLAG_LPWR2RST: Reset due to illegal D2 DSTANDBY flag

Return value:

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

[__HAL_RCC_GET_FLAG](#)

RCC HCLK Clock Source

[RCC_HCLK_DIV1](#)

[RCC_HCLK_DIV2](#)

[RCC_HCLK_DIV4](#)

[RCC_HCLK_DIV8](#)

[RCC_HCLK_DIV16](#)

[RCC_HCLK_DIV64](#)

[RCC_HCLK_DIV128](#)

[RCC_HCLK_DIV256](#)

[RCC_HCLK_DIV512](#)

RCC HSE Config

[RCC_HSE_OFF](#)

[RCC_HSE_ON](#)

RCC_HSE_BYPASS

RCC HSI48 Config

RCC_HSI48_OFF

RCC_HSI48_ON

RCC HSI Config

RCC_HSI_OFF

HSI clock deactivation

RCC_HSI_ON

HSI clock activation

RCC_HSI_DIV1

HSI_DIV1 clock activation

RCC_HSI_DIV2

HSI_DIV2 clock activation

RCC_HSI_DIV4

HSI_DIV4 clock activation

RCC_HSI_DIV8

HSI_DIV8 clock activation

RCC_HSICALIBRATION_DEFAULT

RCC Interrupt

RCC_IT_LSIRDY

RCC_IT_LSERDY

RCC_IT_HSIRDY

RCC_IT_HSERDY

RCC_IT_CSIRDY

RCC_IT_HSI48RDY

RCC_IT_PLLRDY

RCC_IT_PLL2RDY

RCC_IT_PLL3RDY

RCC_IT_LSECSS

RCC_IT_CSS

RCC Private macros to check input parameters

IS_RCC_OSCILLATORTYPE

IS_RCC_HSE

IS_RCC_LSE
IS_RCC_HSI
IS_RCC_HSI48
IS_RCC_LSI
IS_RCC_CSI
IS_RCC_PLL
IS_RCC_PLLSOURCE
IS_RCC_PLLM_VALUE
IS_RCC_PLLN_VALUE
IS_RCC_PLLP_VALUE
IS_RCC_PLLQ_VALUE
IS_RCC_PLLR_VALUE
IS_RCC_PLLCLOCKOUT_VALUE
IS_RCC_CLOCKTYPE
IS_RCC_SYSCLKSOURCE
IS_RCC_SYSCLK
IS_RCC_HCLK
IS_RCC_D1PCLK1
IS_RCC_PCLK1
IS_RCC_PCLK2
IS_RCC_D3PCLK1
IS_RCC_RTCCLKSOURCE
IS_RCC_MCO
IS_RCC_MCO1SOURCE
IS_RCC_MCO2SOURCE
IS_RCC_MCODIV
IS_RCC_FLAG
IS_RCC_CALIBRATION_VALUE

`IS_RCC_CSICALIBRATION_VALUE`

`IS_RCC_STOP_WAKEUPCLOCK`

`IS_RCC_STOP_KERWAKEUPCLOCK`

LSE Drive Config

`RCC_LSEDRIVE_LOW`

LSE low drive capability

`RCC_LSEDRIVE_MEDIUMLOW`

LSE medium low drive capability

`RCC_LSEDRIVE_MEDIUMHIGH`

LSE medium high drive capability

`RCC_LSEDRIVE_HIGH`

LSE high drive capability

RCC LSE Config

`RCC_LSE_OFF`

`RCC_LSE_ON`

`RCC_LSE_BYPASS`

LSE Configuration

`_HAL_RCC_LSE_CONFIG`

Description:

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- `__STATE__`: specifies the new state of the LSE. This parameter can be one of the following values:
 - `RCC_LSE_OFF`: turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
 - `RCC_LSE_ON`: turn ON the LSE oscillator.
 - `RCC_LSE_BYPASS`: LSE oscillator bypassed with external clock.

Notes:

- Transition LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using `HAL_PWR_EnableBkUpAccess()` function before to configure the LSE (to be done once after reset). After enabling the LSE (`RCC_LSE_ON` or `RCC_LSE_BYPASS`), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

RCC LSI Config

`RCC_LSI_OFF`

`RCC_LSI_ON`

RCC MCO1 Clock Source

`RCC_MCO1SOURCE_HSI`

RCC_MCO1SOURCE_LSE

RCC_MCO1SOURCE_HSE

RCC_MCO1SOURCE_PLL1QCLK

RCC_MCO1SOURCE_HSI48

RCC MCO2 Clock Source

RCC_MCO2SOURCE_SYSCLK

RCC_MCO2SOURCE_PLL2PCLK

RCC_MCO2SOURCE_HSE

RCC_MCO2SOURCE_PLLCLK

RCC_MCO2SOURCE_CSICLK

RCC_MCO2SOURCE_LSICLK

RCC MCOx Clock Prescaler

RCC_MCODIV_1

RCC_MCODIV_2

RCC_MCODIV_3

RCC_MCODIV_4

RCC_MCODIV_5

RCC_MCODIV_6

RCC_MCODIV_7

RCC_MCODIV_8

RCC_MCODIV_9

RCC_MCODIV_10

RCC_MCODIV_11

RCC_MCODIV_12

RCC_MCODIV_13

RCC_MCODIV_14

RCC_MCODIV_15

RCC MCO Index

RCC_MCO1

RCC_MCO2

RCC Oscillator Type

RCC_OSCILLATORTYPE_NONE

RCC_OSCILLATORTYPE_HSE

RCC_OSCILLATORTYPE_HSI

RCC_OSCILLATORTYPE_LSE

RCC_OSCILLATORTYPE_LSI

RCC_OSCILLATORTYPE_CSI

RCC_OSCILLATORTYPE_HSI48

RCC PLL1 VCI Range

RCC_PLL1VCIRANGE_0

RCC_PLL1VCIRANGE_1

RCC_PLL1VCIRANGE_2

RCC_PLL1VCIRANGE_3

RCC PLL1 VCO Range

RCC_PLL1VCOWIDE

RCC_PLL1VCOMEDIUM

RCC PLL2 Clock Output

RCC_PLL2_DIVP

RCC_PLL2_DIVQ

RCC_PLL2_DIVR

RCC PLL2 VCI Range

RCC_PLL2VCIRANGE_0

RCC_PLL2VCIRANGE_1

RCC_PLL2VCIRANGE_2

RCC_PLL2VCIRANGE_3

RCC PLL2 VCO Range

RCC_PLL2VCOWIDE

RCC_PLL2VCOMEDIUM

RCC PLL3 Clock Output

RCC_PLL3_DIVP

RCC_PLL3_DIVQ

RCC_PLL3_DIVR

RCC PLL3 VCI Range

RCC_PLL3VCIRANGE_0

RCC_PLL3VCIRANGE_1

RCC_PLL3VCIRANGE_2

RCC_PLL3VCIRANGE_3

RCC PLL3 VCO Range

RCC_PLL3VCOWIDE

RCC_PLL3VCOMEDIUM

RCC PLL Clock Output

RCC_PLL1_DIVP

RCC_PLL1_DIVQ

RCC_PLL1_DIVR

RCC PLL Clock Source

RCC_PLLSOURCE_HSI

RCC_PLLSOURCE_CSI

RCC_PLLSOURCE_HSE

RCC_PLLSOURCE_NONE

RCC PLL Config

RCC_PLL_NONE

RCC_PLL_OFF

RCC_PLL_ON

RCC RTC Clock Source

RCC_RTCCLKSOURCE_LSE

RCC_RTCCLKSOURCE_LSI

RCC_RTCCLKSOURCE_HSE_DIV2

RCC_RTCCLKSOURCE_HSE_DIV3

RCC_RTCCLKSOURCE_HSE_DIV4

RCC_RTCCLKSOURCE_HSE_DIV5

RCC_RTCCLKSOURCE_HSE_DIV6
RCC_RTCCLKSOURCE_HSE_DIV7
RCC_RTCCLKSOURCE_HSE_DIV8
RCC_RTCCLKSOURCE_HSE_DIV9
RCC_RTCCLKSOURCE_HSE_DIV10
RCC_RTCCLKSOURCE_HSE_DIV11
RCC_RTCCLKSOURCE_HSE_DIV12
RCC_RTCCLKSOURCE_HSE_DIV13
RCC_RTCCLKSOURCE_HSE_DIV14
RCC_RTCCLKSOURCE_HSE_DIV15
RCC_RTCCLKSOURCE_HSE_DIV16
RCC_RTCCLKSOURCE_HSE_DIV17
RCC_RTCCLKSOURCE_HSE_DIV18
RCC_RTCCLKSOURCE_HSE_DIV19
RCC_RTCCLKSOURCE_HSE_DIV20
RCC_RTCCLKSOURCE_HSE_DIV21
RCC_RTCCLKSOURCE_HSE_DIV22
RCC_RTCCLKSOURCE_HSE_DIV23
RCC_RTCCLKSOURCE_HSE_DIV24
RCC_RTCCLKSOURCE_HSE_DIV25
RCC_RTCCLKSOURCE_HSE_DIV26
RCC_RTCCLKSOURCE_HSE_DIV27
RCC_RTCCLKSOURCE_HSE_DIV28
RCC_RTCCLKSOURCE_HSE_DIV29
RCC_RTCCLKSOURCE_HSE_DIV30
RCC_RTCCLKSOURCE_HSE_DIV31
RCC_RTCCLKSOURCE_HSE_DIV32
RCC_RTCCLKSOURCE_HSE_DIV33

RCC_RTCCLKSOURCE_HSE_DIV34
RCC_RTCCLKSOURCE_HSE_DIV35
RCC_RTCCLKSOURCE_HSE_DIV36
RCC_RTCCLKSOURCE_HSE_DIV37
RCC_RTCCLKSOURCE_HSE_DIV38
RCC_RTCCLKSOURCE_HSE_DIV39
RCC_RTCCLKSOURCE_HSE_DIV40
RCC_RTCCLKSOURCE_HSE_DIV41
RCC_RTCCLKSOURCE_HSE_DIV42
RCC_RTCCLKSOURCE_HSE_DIV43
RCC_RTCCLKSOURCE_HSE_DIV44
RCC_RTCCLKSOURCE_HSE_DIV45
RCC_RTCCLKSOURCE_HSE_DIV46
RCC_RTCCLKSOURCE_HSE_DIV47
RCC_RTCCLKSOURCE_HSE_DIV48
RCC_RTCCLKSOURCE_HSE_DIV49
RCC_RTCCLKSOURCE_HSE_DIV50
RCC_RTCCLKSOURCE_HSE_DIV51
RCC_RTCCLKSOURCE_HSE_DIV52
RCC_RTCCLKSOURCE_HSE_DIV53
RCC_RTCCLKSOURCE_HSE_DIV54
RCC_RTCCLKSOURCE_HSE_DIV55
RCC_RTCCLKSOURCE_HSE_DIV56
RCC_RTCCLKSOURCE_HSE_DIV57
RCC_RTCCLKSOURCE_HSE_DIV58
RCC_RTCCLKSOURCE_HSE_DIV59
RCC_RTCCLKSOURCE_HSE_DIV60
RCC_RTCCLKSOURCE_HSE_DIV61

RCC_RTCCLKSOURCE_HSE_DIV62

RCC_RTCCLKSOURCE_HSE_DIV63

RCC Stop KernelWakeUpClock

RCC_STOP_KERWAKEUPCLOCK_HSI

RCC_STOP_KERWAKEUPCLOCK_CSI

RCC Stop WakeUpClock

RCC_STOP_WAKEUPCLOCK_HSI

RCC_STOP_WAKEUPCLOCK_CSI

RCC System Clock Source

RCC_SYSCLKSOURCE_CSI

RCC_SYSCLKSOURCE_HSI

RCC_SYSCLKSOURCE_HSE

RCC_SYSCLKSOURCE_PLLCLK

RCC System Clock Type

RCC_CLOCKTYPE_SYSCLK

RCC_CLOCKTYPE_HCLK

RCC_CLOCKTYPE_D1PCLK1

RCC_CLOCKTYPE_PCLK1

RCC_CLOCKTYPE_PCLK2

RCC_CLOCKTYPE_D3PCLK1

RCC SYS Clock Source

RCC_SYSCLK_DIV1

RCC_SYSCLK_DIV2

RCC_SYSCLK_DIV4

RCC_SYSCLK_DIV8

RCC_SYSCLK_DIV16

RCC_SYSCLK_DIV64

RCC_SYSCLK_DIV128

RCC_SYSCLK_DIV256

RCC_SYSCLK_DIV512

56 HAL RCC Extension Driver

56.1 RCCEEx Firmware driver registers structures

56.1.1 RCC_PLL2InitTypeDef

Data Fields

- *uint32_t PLL2M*
- *uint32_t PLL2N*
- *uint32_t PLL2P*
- *uint32_t PLL2Q*
- *uint32_t PLL2R*
- *uint32_t PLL2RGE*
- *uint32_t PLL2VCOSEL*
- *uint32_t PLL2FRACN*

Field Documentation

- *uint32_t RCC_PLL2InitTypeDef::PLL2M*

PLL2M: Division factor for PLL2 VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63

- *uint32_t RCC_PLL2InitTypeDef::PLL2N*

PLL2N: Multiplication factor for PLL2 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512

- *uint32_t RCC_PLL2InitTypeDef::PLL2P*

PLL2P: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowed

- *uint32_t RCC_PLL2InitTypeDef::PLL2Q*

PLL2Q: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- *uint32_t RCC_PLL2InitTypeDef::PLL2R*

PLL2R: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- *uint32_t RCC_PLL2InitTypeDef::PLL2RGE*

PLL2RGE: PLL2 clock Input range This parameter must be a value of **RCC PLL2 VCI Range**

- *uint32_t RCC_PLL2InitTypeDef::PLL2VCOSEL*

PLL2VCOSEL: PLL2 clock Output range This parameter must be a value of **RCC PLL2 VCO Range**

- *uint32_t RCC_PLL2InitTypeDef::PLL2FRACN*

PLL2FRACN: Specifies Fractional Part Of The Multiplication Factor for PLL2 VCO It should be a value between 0 and 8191

56.1.2 RCC_PLL3InitTypeDef

Data Fields

- *uint32_t PLL3M*
- *uint32_t PLL3N*

- `uint32_t PLL3P`
- `uint32_t PLL3Q`
- `uint32_t PLL3R`
- `uint32_t PLL3RGE`
- `uint32_t PLL3VCOSEL`
- `uint32_t PLL3FRACN`

Field Documentation

- `uint32_t RCC_PLL3InitTypeDef::PLL3M`

PLL3M: Division factor for PLL3 VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63

- `uint32_t RCC_PLL3InitTypeDef::PLL3N`

PLL3N: Multiplication factor for PLL3 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512

- `uint32_t RCC_PLL3InitTypeDef::PLL3P`

PLL3P: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 odd division factors are not allowed

- `uint32_t RCC_PLL3InitTypeDef::PLL3Q`

PLL3Q: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- `uint32_t RCC_PLL3InitTypeDef::PLL3R`

PLL3R: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- `uint32_t RCC_PLL3InitTypeDef::PLL3RGE`

PLL3RGE: PLL3 clock Input range This parameter must be a value of **RCC PLL3 VCI Range**

- `uint32_t RCC_PLL3InitTypeDef::PLL3VCOSEL`

PLL3VCOSEL: PLL3 clock Output range This parameter must be a value of **RCC PLL3 VCO Range**

- `uint32_t RCC_PLL3InitTypeDef::PLL3FRACN`

PLL3FRACN: Specifies Fractional Part Of The Multiplication Factor for PLL3 VCO It should be a value between 0 and 8191

56.1.3 PLL1_ClocksTypeDef

Data Fields

- `uint32_t PLL1_P_Frequency`
- `uint32_t PLL1_Q_Frequency`
- `uint32_t PLL1_R_Frequency`

Field Documentation

- `uint32_t PLL1_ClocksTypeDef::PLL1_P_Frequency`

- `uint32_t PLL1_ClocksTypeDef::PLL1_Q_Frequency`

- `uint32_t PLL1_ClocksTypeDef::PLL1_R_Frequency`

56.1.4 PLL2_ClocksTypeDef

Data Fields

- `uint32_t PLL2_P_Frequency`

- `uint32_t PLL2_Q_Frequency`
- `uint32_t PLL2_R_Frequency`

Field Documentation

- `uint32_t PLL2_ClocksTypeDef::PLL2_P_Frequency`
- `uint32_t PLL2_ClocksTypeDef::PLL2_Q_Frequency`
- `uint32_t PLL2_ClocksTypeDef::PLL2_R_Frequency`

56.1.5 PLL3_ClocksTypeDef

Data Fields

- `uint32_t PLL3_P_Frequency`
- `uint32_t PLL3_Q_Frequency`
- `uint32_t PLL3_R_Frequency`

Field Documentation

- `uint32_t PLL3_ClocksTypeDef::PLL3_P_Frequency`
- `uint32_t PLL3_ClocksTypeDef::PLL3_Q_Frequency`
- `uint32_t PLL3_ClocksTypeDef::PLL3_R_Frequency`

56.1.6 RCC_PeriphCLKInitTypeDef

Data Fields

- `uint32_t PeriphClockSelection`
- `RCC_PLL2InitTypeDef PLL2`
- `RCC_PLL3InitTypeDef PLL3`
- `uint32_t FmcClockSelection`
- `uint32_t QspiClockSelection`
- `uint32_t SdmmcClockSelection`
- `uint32_t CkperClockSelection`
- `uint32_t Sai1ClockSelection`
- `uint32_t Sai23ClockSelection`
- `uint32_t Spi123ClockSelection`
- `uint32_t Spi45ClockSelection`
- `uint32_t SpdifrxClockSelection`
- `uint32_t Dfsm1ClockSelection`
- `uint32_t FdcanClockSelection`
- `uint32_t Swpmi1ClockSelection`
- `uint32_t Usart234578ClockSelection`
- `uint32_t Usart16ClockSelection`
- `uint32_t RngClockSelection`
- `uint32_t I2c123ClockSelection`
- `uint32_t UsbClockSelection`
- `uint32_t CecClockSelection`
- `uint32_t Loptim1ClockSelection`
- `uint32_t Lpuart1ClockSelection`
- `uint32_t I2c4ClockSelection`

- `uint32_t Lptim2ClockSelection`
- `uint32_t Lptim345ClockSelection`
- `uint32_t AdcClockSelection`
- `uint32_t Sai4AClockSelection`
- `uint32_t Sai4BClockSelection`
- `uint32_t Spi6ClockSelection`
- `uint32_t RTCClockSelection`
- `uint32_t Hrtim1ClockSelection`
- `uint32_t TIMPresSelection`

Field Documentation

- `uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection`

The Extended Clock to be configured. This parameter can be a value of **RCCEEx Periph Clock Selection**

- `RCC_PLL2InitTypeDef RCC_PeriphCLKInitTypeDef::PLL2`

PLL2structure parameters. This parameter will be used only when PLL2 is selected as kernel clock Source for some peripherals

- `RCC_PLL3InitTypeDef RCC_PeriphCLKInitTypeDef::PLL3`

PLL3 structure parameters. This parameter will be used only when PLL2 is selected as kernel clock Source for some peripherals

- `uint32_t RCC_PeriphCLKInitTypeDef::FmcClockSelection`

Specifies FMC clock source This parameter can be a value of **RCCEEx FMC Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::QspiClockSelection`

Specifies QSPI clock source This parameter can be a value of **RCCEEx QSPI Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::SdmmcClockSelection`

Specifies SDMMC clock source This parameter can be a value of **RCCEEx SDMMC Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::CkperClockSelection`

Specifies CKPER clock source This parameter can be a value of **RCCEEx CLKP Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::Sai1ClockSelection`

Specifies SAI1 clock source This parameter can be a value of **SAI1 Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::Sai23ClockSelection`

Specifies SAI2/3 clock source This parameter can be a value of **SAI2/3 Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::Spi123ClockSelection`

Specifies SPI1/2/3 clock source This parameter can be a value of **SPI1/2/3 Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::Spi45ClockSelection`

Specifies SPI4/5 clock source This parameter can be a value of **SPI4/5 Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::SpdifrxClockSelection`

Specifies SPDIFRX Clock clock source This parameter can be a value of **RCCEEx SPDIFRX Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::Dfsm1ClockSelection`

Specifies DFSDM1 Clock clock source This parameter can be a value of **RCCEEx DFSDM1 Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::FdcanClockSelection`

Specifies FDCAN Clock clock source This parameter can be a value of **RCCEEx FDCAN Clock Source**

- `uint32_t RCC_PeriphCLKInitTypeDef::Swpmi1ClockSelection`

Specifies SWPMI1 Clock clock source This parameter can be a value of **RCCEEx SWPMI1 Clock Source**

- **`uint32_t RCC_PeriphCLKInitTypeDef::Usart234578ClockSelection`**
Specifies USART2/3/4/5/7/8 clock source This parameter can be a value of **RCCEEx USART2/3/4/5/7/8 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Usart16ClockSelection`**
Specifies USART1/6 clock source This parameter can be a value of **RCCEEx USART1/6 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::RngClockSelection`**
Specifies RNG clock source This parameter can be a value of **RCCEEx RNG Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c123ClockSelection`**
Specifies I2C1/2/3 clock source This parameter can be a value of **RCCEEx I2C1/2/3 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection`**
Specifies USB clock source This parameter can be a value of **Section 56.3.1 RCCEEx**
- **`uint32_t RCC_PeriphCLKInitTypeDef::CecClockSelection`**
Specifies CEC clock source This parameter can be a value of **RCCEEx CEC Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection`**
Specifies LPTIM1 clock source This parameter can be a value of **RCCEEx LPTIM1 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection`**
Specifies LPUART1 clock source This parameter can be a value of **RCCEEx LPUART1 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection`**
Specifies I2C4 clock source This parameter can be a value of **RCCEEx I2C4 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Lptim2ClockSelection`**
Specifies LPTIM2 clock source This parameter can be a value of **RCCEEx LPTIM2 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Lptim345ClockSelection`**
Specifies LPTIM3/4/5 clock source This parameter can be a value of **RCCEEx LPTIM3/4/5 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::AdcClockSelection`**
Specifies ADC interface clock source This parameter can be a value of **RCCEEx ADC Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Sai4AClockSelection`**
Specifies SAI4A clock source This parameter can be a value of **SAI4A Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Sai4BClockSelection`**
Specifies SAI4B clock source This parameter can be a value of **SAI4B Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Spi6ClockSelection`**
Specifies SPI6 clock source This parameter can be a value of **SPI6 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection`**
Specifies RTC Clock clock source This parameter can be a value of **RCC RTC Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Hrtim1ClockSelection`**
Specifies HRTIM1 Clock clock source This parameter can be a value of **RCC Extended HRTIM1 Clock Source**
- **`uint32_t RCC_PeriphCLKInitTypeDef::TIMPresSelection`**
Specifies TIM Clock Prescalers Selection. This parameter can be a value of **RCCEEx TIM Prescaler Selection**

56.1.7 RCC_CRSInitTypeDef

Data Fields

- `uint32_t Prescaler`
- `uint32_t Source`
- `uint32_t Polarity`
- `uint32_t ReloadValue`
- `uint32_t ErrorLimitValue`
- `uint32_t HSI48CalibrationValue`

Field Documentation

- `uint32_t RCC_CRSInitTypeDef::Prescaler`
Specifies the division factor of the SYNC signal. This parameter can be a value of **RCCEEx CRS SynchroDivider**
- `uint32_t RCC_CRSInitTypeDef::Source`
Specifies the SYNC signal source. This parameter can be a value of **RCCEEx CRS SynchroSource**
- `uint32_t RCC_CRSInitTypeDef::Polarity`
Specifies the input polarity for the SYNC signal source. This parameter can be a value of **RCCEEx CRS SynchroPolarity**
- `uint32_t RCC_CRSInitTypeDef::ReloadValue`
Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro `__HAL_RCC_CRS_RELOADVALUE_CALCULATE(__FTARGET__, __FSYNC__)` This parameter must be a number between 0 and 0xFFFF or a value of **RCCEEx CRS ReloadValueDefault**.
- `uint32_t RCC_CRSInitTypeDef::ErrorLimitValue`
Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of **RCCEEx CRS ErrorLimitDefault**
- `uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue`
Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x3F or a value of **RCCEEx CRS HSI48CalibrationDefault**

56.1.8 RCC_CRSSynchroInfoTypeDef

Data Fields

- `uint32_t ReloadValue`
- `uint32_t HSI48CalibrationValue`
- `uint32_t FreqErrorCapture`
- `uint32_t FreqErrorDirection`

Field Documentation

- `uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue`
Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- `uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue`
Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F
- `uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture`
Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- `uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection`

Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of **RCCEx CRS FreqErrorDirection**

56.2 RCCEx Firmware driver API description

56.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

Note: *Important note: Care must be taken when HAL_RCCEEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.*

This section contains the following APIs:

- [HAL_RCCEEx_PeriphCLKConfig](#)
- [HAL_RCCEEx_GetPeriphCLKConfig](#)
- [HAL_RCCEEx_GetPeriphCLKFreq](#)
- [HAL_RCCEEx_GetD1PCLK1Freq](#)
- [HAL_RCCEEx_GetD3PCLK1Freq](#)
- [HAL_RCCEEx_GetPLL2ClockFreq](#)
- [HAL_RCCEEx_GetPLL3ClockFreq](#)
- [HAL_RCCEEx_GetPLL1ClockFreq](#)
- [HAL_RCCEEx_GetD1SysClockFreq](#)
- [HAL_RCCEEx_EnableLSECSS](#)
- [HAL_RCCEEx_DisableLSECSS](#)
- [HAL_RCCEEx_WakeUpStopCLKConfig](#)
- [HAL_RCCEEx_KerWakeUpStopCLKConfig](#)
- [HAL_RCCEEx_WWDGxSysResetConfig](#)

56.2.2 Detailed description of functions

HAL_RCCEEx_PeriphCLKConfig

Function name

HAL_StatusTypeDef HAL_RCCEEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks(SDMMC, CKPER, FMC, QSPI, DSI, SPI45, SPDIF, DFSDM1, FDCAN, SWPMI, SAI23, SAI1, SPI123, USART234578, USART16, RNG, HRTIM1, I2C123, USB, CEC, LPTIM1, LPUART1, I2C4, LPTIM2, LPTIM345, ADC, SAI4A, SAI4B, SPI6, RTC).

Return values

- **HAL:** status

Notes

- Care must be taken when HAL_RCCEEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.

HAL_RCCEEx_GetPeriphCLKConfig

Function name

```
void HAL_RCCEEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
```

Function description

Get the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks : (SDMMC, CKPER, FMC, QSPI, DSI, SPI45, SPDIF, DFSDM1, FDCAN, SWPML, SAI23, SAI1, SPI123, USART234578, USART16, RNG, HRTIM1, I2C123, USB, CEC, LPTIM1, LPUART1, I2C4, LPTIM2, LPTIM345, ADC, SAI4A, SAI4B, SPI6, RTC, TIM).

Return values

- **None:**

HAL_RCCEEx_GetPeriphCLKFreq

Function name

```
uint32_t HAL_RCCEEx_GetPeriphCLKFreq (uint32_t PeriphClk)
```

Function description

Return the peripheral clock frequency for a given peripheral(SAI..)

Parameters

- **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values:
 - RCC_PERIPHCLK_SAI1 : SAI1 peripheral clock
 - RCC_PERIPHCLK_SAI23 : SAI2/3 peripheral clock
 - RCC_PERIPHCLK_SAI4A : SAI4A peripheral clock
 - RCC_PERIPHCLK_SAI4B : SAI4B peripheral clock
 - RCC_PERIPHCLK_SPI123: SPI1/2/3 peripheral clock

Return values

- **Frequency:** in KHz

Notes

- Return 0 if peripheral clock identifier not managed by this API

HAL_RCCEEx_GetD1PCLK1Freq

Function name

```
uint32_t HAL_RCCEEx_GetD1PCLK1Freq (void )
```

Function description

Returns the D1PCLK1 frequency.

Return values

- **D1PCLK1:** frequency

Notes

- Each time D1PCLK1 changes, this function must be called to update the right D1PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEEx_GetD3PCLK1Freq

Function name

```
uint32_t HAL_RCCEEx_GetD3PCLK1Freq (void )
```

Function description

Returns the D3PCLK1 frequency.

Return values

- **D3PCLK1:** frequency

Notes

- Each time D3PCLK1 changes, this function must be called to update the right D3PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEEx_GetD1SysClockFreq

Function name

```
uint32_t HAL_RCCEEx_GetD1SysClockFreq (void )
```

Function description

Returns the main Core frequency.

Return values

- **HCLK:** frequency

Notes

- Each time core clock changes, this function must be called to update the right system core clock value. Otherwise, any configuration based on this function will be incorrect.
- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

HAL_RCCEEx_GetPLL1ClockFreq

Function name

```
void HAL_RCCEEx_GetPLL1ClockFreq (PLL1_ClocksTypeDef * PLL1_Clocks)
```

Function description

Returns the PLL1 clock frequencies :PLL1_P_Frequency,PLL1_R_Frequency and PLL1_Q_Frequency.

Parameters

- **PLL1_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL1 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL1CLK changes, this function must be called to update the right PLL1CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEEx_GetPLL2ClockFreq

Function name

```
void HAL_RCCEEx_GetPLL2ClockFreq (PLL2_ClocksTypeDef * PLL2_Clocks)
```

Function description

Returns the PLL2 clock frequencies :PLL2_P_Frequency,PLL2_R_Frequency and PLL2_Q_Frequency.

Parameters

- **PLL2_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL2 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL2CLK changes, this function must be called to update the right PLL2CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEEx_GetPLL3ClockFreq

Function name

```
void HAL_RCCEEx_GetPLL3ClockFreq (PLL3_ClocksTypeDef * PLL3_Clocks)
```

Function description

Returns the PLL3 clock frequencies :PLL3_P_Frequency,PLL3_R_Frequency and PLL3_Q_Frequency.

Parameters

- **PLL3_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL3 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL3CLK changes, this function must be called to update the right PLL3CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEEx_WakeUpStopCLKConfig

Function name

```
void HAL_RCCEEx_WakeUpStopCLKConfig (uint32_t WakeUpClk)
```

Function description

Configure the oscillator clock source for wakeup from Stop and CSS backup clock.

Parameters

- **WakeUpClk:** Wakeup clock This parameter can be one of the following values:
 - RCC_STOP_WAKEUPCLOCK_CSI: CSI oscillator selection
 - RCC_STOP_WAKEUPCLOCK_HSI: HSI oscillator selection

Return values

- **None:**

Notes

- This function shall not be called after the Clock Security System on HSE has been enabled.

HAL_RCCEEx_KerWakeUpStopCLKConfig

Function name

```
void HAL_RCCEEx_KerWakeUpStopCLKConfig (uint32_t WakeUpClk)
```

Function description

Configure the oscillator Kernel clock source for wakeup from Stop.

Parameters

- **WakeUpClk:** Kernel Wakeup clock This parameter can be one of the following values:
 - RCC_STOP_KERWAKEUPCLOCK_CSI: CSI oscillator selection
 - RCC_STOP_KERWAKEUPCLOCK_HSI: HSI oscillator selection

Return values

- **None:**

HAL_RCCEEx_EnableLSECSS

Function name

```
void HAL_RCCEEx_EnableLSECSS (void )
```

Function description

Enables the LSE Clock Security System.

Return values

- **None:**

Notes

- Prior to enable the LSE Clock Security System, LSE oscillator is to be enabled with HAL_RCC_OscConfig() and the LSE oscillator clock is to be selected as RTC clock with HAL_RCCEEx_PeriphCLKConfig().

HAL_RCCEEx_DisableLSECSS

Function name

```
void HAL_RCCEEx_DisableLSECSS (void )
```

Function description

Disables the LSE Clock Security System.

Return values

- **None:**

Notes

- LSE Clock Security System can only be disabled after a LSE failure detection.

HAL_RCCEEx_WWDGxSysResetConfig

Function name

```
void HAL_RCCEEx_WWDGxSysResetConfig (uint32_t RCC_WWDGx)
```

Function description

Configure WWDG1 to generate a system reset not only CPU reset(default) when a time-out occurs.

Parameters

- **RCC_WWDGx:** WWDGx to be configured This parameter can be one of the following values:
 - RCC_WWDG1: WWDG1 generates system reset

Return values

- **None:**

Notes

- This bit can be set by software but is cleared by hardware during a system reset

HAL_RCCEEx_CRSConfig

Function name

```
void HAL_RCCEEx_CRSConfig (RCC_CRSInitTypeDef * pInit)
```

Function description

Start automatic synchronization for polling mode.

Parameters

- **pInit:** Pointer on RCC_CRSInitTypeDef structure

Return values

- **None:**

HAL_RCCEEx_CRSSoftwareSynchronizationGenerate

Function name

```
void HAL_RCCEEx_CRSSoftwareSynchronizationGenerate (void )
```

Function description

Generate the software synchronization event.

Return values

- **None:**

HAL_RCCEEx_CRSGetSynchronizationInfo

Function name

```
void HAL_RCCEEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)
```

Function description

Return synchronization info.

Parameters

- **pSynchroInfo:** Pointer on RCC_CRSSynchroInfoTypeDef structure

Return values

- **None:**

HAL_RCCEEx_CRSWaitSynchronization

Function name

```
uint32_t HAL_RCCEEx_CRSWaitSynchronization (uint32_t Timeout)
```

Function description

Wait for CRS Synchronization status.

Parameters

- **Timeout:** Duration of the time-out

Return values

- **Combination:** of Synchronization status This parameter can be a combination of the following values:
 - RCC_CRS_TIMEOUT
 - RCC_CRS_SYNCOK
 - RCC_CRS_SYNCWARN
 - RCC_CRS_SYNCERR
 - RCC_CRS_SYNCMISS
 - RCC_CRS_TRIMOVF

Notes

- Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.
- If Time-out set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

HAL_RCCEEx_CRS_IRQHandler

Function name

```
void HAL_RCCEEx_CRS_IRQHandler (void )
```

Function description

Handle the Clock Recovery System interrupt request.

Return values

- **None:**

HAL_RCCEEx_CRS_SyncOkCallback

Function name

```
void HAL_RCCEEx_CRS_SyncOkCallback (void )
```

Function description

RCCEx Clock Recovery System SYNCOK interrupt callback.

Return values

- **none:**

HAL_RCCEEx_CRS_SyncWarnCallback

Function name

```
void HAL_RCCEEx_CRS_SyncWarnCallback (void )
```

Function description

RCCEx Clock Recovery System SYNCWARN interrupt callback.

Return values

- **none:**

HAL_RCCEEx_CRS_ExpectedSyncCallback

Function name

void HAL_RCCEEx_CRS_ExpectedSyncCallback (void)

Function description

RCCEEx Clock Recovery System Expected SYNC interrupt callback.

Return values

- **none:**

HAL_RCCEEx_CRS_ErrorCallback

Function name

void HAL_RCCEEx_CRS_ErrorCallback (uint32_t Error)

Function description

RCCEEx Clock Recovery System Error interrupt callback.

Parameters

- **Error:** Combination of Error status. This parameter can be a combination of the following values:
 - RCC_CRS_SYNCERR
 - RCC_CRS_SYNCMISS
 - RCC_CRS_TRIMOVF

Return values

- **none:**

56.3 RCCEEx Firmware driver defines

56.3.1 RCCEEx

RCCEEx ADC Clock Source

RCC_ADCCLKSOURCE_PLL2

RCC_ADCCLKSOURCE_PLL3

RCC_ADCCLKSOURCE_CLKP

RCCEEx CEC Clock Source

RCC_CECCLKSOURCE_LSE

RCC_CECCLKSOURCE_LSI

RCC_CECCLKSOURCE_CSI

RCCEEx CLKP Clock Source

RCC_CLKPSOURCE_HSI

RCC_CLKPSOURCE_CSI

RCC_CLKPSOURCE_HSE

RCCEEx CRS ErrorLimitDefault

RCC_CRS_ERRORLIMIT_DEFAULT

Default Frequency error limit

RCCEEx CRS Extended Features

_HAL_RCC_CRS_FREQ_ERROR_COUNTER_ENABLE

Description:

- Enable the oscillator clock for frequency error counter.

Return value:

- None

Notes:

- when the CEN bit is set the CRS_CFG register becomes write-protected.

_HAL_RCC_CRS_FREQ_ERROR_COUNTER_DISABLE

Description:

- Disable the oscillator clock for frequency error counter.

Return value:

- None

_HAL_RCC_CRS_AUTOMATIC_CALIB_ENABLE

Description:

- Enable the automatic hardware adjustment of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFG register becomes write-protected.

_HAL_RCC_CRS_AUTOMATIC_CALIB_DISABLE

Description:

- Enable or disable the automatic hardware adjustment of TRIM bits.

Return value:

- None

_HAL_RCC_CRS_RELOADVALUE_CALCULATE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- _FTARGET_: Target frequency (value in Hz)
- _FSYNC_: Synchronization signal frequency (value in Hz)

Return value:

- None

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after pre-scaling. It is then decreased by one in order to reach the

expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

RCCEEx CRS Flags

RCC_CRS_FLAG_SYNCOK

SYNC event OK flag

RCC_CRS_FLAG_SYNCWARN

SYNC warning flag

RCC_CRS_FLAG_ERR

Error flag

RCC_CRS_FLAG_ESYNC

Expected SYNC flag

RCC_CRS_FLAG_SYNCERR

SYNC error

RCC_CRS_FLAG_SYNCMISS

SYNC missed

RCC_CRS_FLAG_TRIMOVF

Trimming overflow or underflow

RCCEEx CRS FreqErrorDirection

RCC_CRS_FREQERRORDIR_UP

Upcounting direction, the actual frequency is above the target

RCC_CRS_FREQERRORDIR_DOWN

Downcounting direction, the actual frequency is below the target

RCCEEx CRS HSI48CalibrationDefault

RCC_CRS_HSI48CALIBRATION_DEFAULT

The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

RCCEEx CRS Interrupt Sources

RCC_CRS_IT_SYNCOK

SYNC event OK

RCC_CRS_IT_SYNCWARN

SYNC warning

RCC_CRS_IT_ERR

Error

RCC_CRS_IT_ESYNC

Expected SYNC

RCC_CRS_IT_SYNCERR

SYNC error

RCC_CRS_IT_SYNCMISS

SYNC missed

RCC_CRS_IT_TRIMOVF

Trimming overflow or underflow

RCCEEx CRS ReloadValueDefault**RCC_CRS_RELOADVALUE_DEFAULT**

The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

RCCEEx CRS Status**RCC_CRS_NONE****RCC_CRS_TIMEOUT****RCC_CRS_SYNCOK****RCC_CRS_SYNCWARN****RCC_CRS_SYNCERR****RCC_CRS_SYNCMISS****RCC_CRS_TRIMOVF*****RCCEEx CRS SynchroDivider*****RCC_CRS_SYNC_DIV1**

Synchro Signal not divided (default)

RCC_CRS_SYNC_DIV2

Synchro Signal divided by 2

RCC_CRS_SYNC_DIV4

Synchro Signal divided by 4

RCC_CRS_SYNC_DIV8

Synchro Signal divided by 8

RCC_CRS_SYNC_DIV16

Synchro Signal divided by 16

RCC_CRS_SYNC_DIV32

Synchro Signal divided by 32

RCC_CRS_SYNC_DIV64

Synchro Signal divided by 64

RCC_CRS_SYNC_DIV128

Synchro Signal divided by 128

RCCEEx CRS SynchroPolarity**RCC_CRS_SYNC_POLARITY_RISING**

Synchro Active on rising edge (default)

RCC_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

RCCEEx CRS SynchroSource**RCC_CRS_SYNC_SOURCE_USB2**

Synchro Signal source USB2 SOF

RCC_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

RCC_CRS_SYNC_SOURCE_USB1

Synchro Signal source USB1 SOF (default)

RCCEEx DFSDM1 Clock Source**RCC_DFSDM1CLKSOURCE_D2PCLK1****RCC_DFSDM1CLKSOURCE_SYS*****RCCEEx Exported Constants*****RCC_USBCLKSOURCE_PLL****RCC_USBCLKSOURCE_PLL****RCC_USBCLKSOURCE_PLL3****RCC_USBCLKSOURCE_PLL3****RCC_USBCLKSOURCE_HSI48****RCC_USBCLKSOURCE_HSI48*****RCCEEx Exported Macros*****_HAL_RCC_PLL2_ENABLE****Notes:**

- After enabling PLL2, the application software should wait on PLL2RDY flag to be set indicating that PLL2 clock is stable and can be used as kernel clock source. PLL2 is disabled by hardware when entering STOP and STANDBY modes.

_HAL_RCC_PLL2_DISABLE**_HAL_RCC_PLL2CLKOUT_ENABLE****Description:**

- Enables or disables each clock output (PLL2_P_CLK, PLL2_Q_CLK, PLL2_R_CLK)

Parameters:

- RCC_PLL2ClockOut: Specifies the PLL2 clock to be outputted This parameter can be one of the following values:
 - RCC_PLL2_DIVP: This clock is used to generate system clock (up to 400MHZ)
 - RCC_PLL2_DIVQ: This clock is used to generate peripherals clock (up to 400MHZ)
 - RCC_PLL2_DIVR: This clock is used to generate peripherals clock (up to 400MHZ)

Return value:

- None

Notes:

- Enabling/disabling Those Clocks can be any time without the need to stop the PLL2, This is mainly used to save Power.

[__HAL_RCC_PLL2CLKOUT_DISABLE](#)[__HAL_RCC_PLL2FRACN_ENABLE](#)**Description:**

- Enables or disables Fractional Part Of The Multiplication Factor of PLL2 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be any time without the need to stop the PLL2

[__HAL_RCC_PLL2FRACN_DISABLE](#)[__HAL_RCC_PLL2_CONFIG](#)**Description:**

- Macro to configures the PLL2 multiplication and division factors.

Parameters:

- PLL2M: specifies the division factor for PLL2 VCO input clock This parameter must be a number between 1 and 63.
- PLL2N: specifies the multiplication factor for PLL2 VCO output clock This parameter must be a number between 4 and 512.
- PLL2P: specifies the division factor for peripheral kernel clocks This parameter must be a number between 2 and 128 (where odd numbers not allowed)
- PLL2Q: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- PLL2R: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None

Notes:

- This function must be used only when PLL2 is disabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 16 MHz.
- You have to set the PLL2N parameter correctly to ensure that the VCO output frequency is between 150 and 420 MHz (when in medium VCO range) or between 192 and 836 MHZ (when in wide VCO range)

[__HAL_RCC_PLL2FRACN_CONFIG](#)**Description:**

- Macro to configures PLL2 clock Fractional Part Of The Multiplication Factor.

Parameters:

- RCC_PLL2FRACN: Specifies Fractional Part Of The Multiplication factor for PLL2 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL2 VCO
- Warning: the software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL2VCOSEL = 0 150 to 420 MHz if PLL2VCOSEL = 1.

__HAL_RCC_PLL2_VCIRANGE

Description:

- Macro to select the PLL2 reference frequency range.

Parameters:

- __RCC_PLL2VCIRange__: specifies the PLL2 input frequency range This parameter can be one of the following values:
 - RCC_PLL2VCIRANGE_0: Range frequency is between 1 and 2 MHz
 - RCC_PLL2VCIRANGE_1: Range frequency is between 2 and 4 MHz
 - RCC_PLL2VCIRANGE_2: Range frequency is between 4 and 8 MHz
 - RCC_PLL2VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL2_VCORANGE

Description:

- Macro to select the PLL2 reference frequency range.

Parameters:

- __RCC_PLL2VCORange__: Specifies the PLL2 input frequency range This parameter can be one of the following values:
 - RCC_PLL2VCOWIDE: Range frequency is between 192 and 836 MHz
 - RCC_PLL2VCOMEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

__HAL_RCC_PLL3_ENABLE

Notes:

- After enabling PLL3, the application software should wait on PLL3RDY flag to be set indicating that PLL3 clock is stable and can be used as kernel clock source. PLL3 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL3_DISABLE

__HAL_RCC_PLL3FRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL3 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be any time without the need to stop the PLL3

__HAL_RCC_PLL3FRACN_DISABLE

__HAL_RCC_PLL3CLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL3_P_CLK, PLL3_Q_CLK, PLL3_R_CLK)

Parameters:

- `__RCC_PLL3ClockOut__`: specifies the PLL3 clock to be outputted This parameter can be one of the following values:
 - `RCC_PLL3_DIVP`: This clock is used to generate system clock (up to 400MHZ)
 - `RCC_PLL3_DIVQ`: This clock is used to generate peripherals clock (up to 400MHZ)
 - `RCC_PLL3_DIVR`: This clock is used to generate peripherals clock (up to 400MHZ)

Return value:

- None

Notes:

- Enabling/disabling Those Clocks can be any time without the need to stop the PLL3, This is mainly used to save Power.

[_HAL_RCC_PLL3CLKOUT_DISABLE](#)

[_HAL_RCC_PLL3_CONFIG](#)

Description:

- Macro to configures the PLL3 multiplication and division factors.

Parameters:

- `__PLL3M__`: specifies the division factor for PLL3 VCO input clock This parameter must be a number between 1 and 63.
- `__PLL3N__`: specifies the multiplication factor for PLL3 VCO output clock This parameter must be a number between 4 and 512.
- `__PLL3P__`: specifies the division factor for peripheral kernel clocks This parameter must be a number between 2 and 128 (where odd numbers not allowed)
- `__PLL3Q__`: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- `__PLL3R__`: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None

Notes:

- This function must be used only when PLL3 is disabled.
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 16 MHz.
- You have to set the PLL3N parameter correctly to ensure that the VCO output frequency is between 150 and 420 MHz (when in medium VCO range) or between 192 and 836 MHZ (when in wide VCO range)

[_HAL_RCC_PLL3FRACN_CONFIG](#)

Description:

- Macro to configures PLL3 clock Fractional Part of The Multiplication Factor.

Parameters:

- `__RCC_PLL3FRACN__`: specifies Fractional Part Of The Multiplication Factor for PLL3 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL3 VCO
- Warning: the software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL3VCOSEL = 0 150 to 420 MHz if PLL3VCOSEL = 1.

__HAL_RCC_PLL3_VCIRANGE

Description:

- Macro to select the PLL3 reference frequency range.

Parameters:

- RCC_PLL3VCIRange: specifies the PLL1 input frequency range This parameter can be one of the following values:
 - RCC_PLL3VCIRANGE_0: Range frequency is between 1 and 2 MHz
 - RCC_PLL3VCIRANGE_1: Range frequency is between 2 and 4 MHz
 - RCC_PLL3VCIRANGE_2: Range frequency is between 4 and 8 MHz
 - RCC_PLL3VCIRANGE_3: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL3_VCORANGE

Description:

- Macro to select the PLL3 reference frequency range.

Parameters:

- RCC_PLL3VCORange: specifies the PLL1 input frequency range This parameter can be one of the following values:
 - RCC_PLL3VCOWIDE: Range frequency is between 192 and 836 MHz
 - RCC_PLL3VCOMEDIUM: Range frequency is between 150 and 420 MHz

Return value:

- None

__HAL_RCC_SAI1_CONFIG

Description:

- Macro to Configure the SAI1 clock source.

Parameters:

- RCC_SAI1CLKSource: defines the SAI1 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SAI1CLKSOURCE_PLL: SAI1 clock = PLL
 - RCC_SAI1CLKSOURCE_PLL2: SAI1 clock = PLL2
 - RCC_SAI1CLKSOURCE_PLL3: SAI1 clock = PLL3
 - RCC_SAI1CLKSOURCE_OSC: SAI1 clock = OSC
 - RCC_SAI1CLKSOURCE_PIN: SAI1 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI1_SOURCE

Description:

- Macro to get the SAI1 clock source.

Return value:

- The: clock source can be one of the following values:

- RCC_SAI1CLKSOURCE_PLL: SAI1 clock = PLL
- RCC_SAI1CLKSOURCE_PLL2: SAI1 clock = PLL2
- RCC_SAI1CLKSOURCE_PLL3: SAI1 clock = PLL3
- RCC_SAI1CLKSOURCE_CLKP: SAI1 clock = CLKP
- RCC_SAI1CLKSOURCE_PIN: SAI1 clock = External Clock

__HAL_RCC_SPDIFRX_CONFIG

Description:

- Macro to Configure the SPDIFRX clock source.

Parameters:

- RCC_SPDIFCLKSource: defines the SPDIFRX clock source. This clock is derived from system PLL, PLL2, PLL3, or internal OSC clock This parameter can be one of the following values:
 - RCC_SPDIFRXCLKSOURCE_PLL: SPDIFRX clock = PLL
 - RCC_SPDIFRXCLKSOURCE_PLL2: SPDIFRX clock = PLL2
 - RCC_SPDIFRXCLKSOURCE_PLL3: SPDIFRX clock = PLL3
 - RCC_SPDIFRXCLKSOURCE_HSI: SPDIFRX clock = HSI

Return value:

- None

__HAL_RCC_GET_SPDIFRX_SOURCE

Description:

- Macro to get the SPDIFRX clock source.

Return value:

- None

__HAL_RCC_SAI23_CONFIG

Description:

- Macro to Configure the SAI2/3 clock source.

Parameters:

- RCC_SAI23CLKSource: defines the SAI2/3 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SAI23CLKSOURCE_PLL: SAI2/3 clock = PLL
 - RCC_SAI23CLKSOURCE_PLL2: SAI2/3 clock = PLL2
 - RCC_SAI23CLKSOURCE_PLL3: SAI2/3 clock = PLL3
 - RCC_SAI23CLKSOURCE_CLKP: SAI2/3 clock = CLKP
 - RCC_SAI23CLKSOURCE_PIN: SAI2/3 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI23_SOURCE

Description:

- Macro to get the SAI2/3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SAI23CLKSOURCE_PLL: SAI2/3 clock = PLL
 - RCC_SAI23CLKSOURCE_PLL2: SAI2/3 clock = PLL2
 - RCC_SAI23CLKSOURCE_PLL3: SAI2/3 clock = PLL3

- RCC_SAI23CLKSOURCE_CLKP: SAI2/3 clock = CLKp
- RCC_SAI23CLKSOURCE_PIN: SAI2/3 clock = External Clock

__HAL_RCC_SAI2_CONFIG

Description:

- Macro to Configure the SAI2 clock source.

Parameters:

- __RCC_SAI2CLKSource__: defines the SAI2 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SAI2CLKSOURCE_PLL: SAI2 clock = PLL
 - RCC_SAI2CLKSOURCE_PLL2: SAI2 clock = PLL2
 - RCC_SAI2CLKSOURCE_PLL3: SAI2 clock = PLL3
 - RCC_SAI2CLKSOURCE_CLKP: SAI2 clock = CLKp
 - RCC_SAI2CLKSOURCE_PIN: SAI2 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI2_SOURCE

Description:

- Macro to get the SAI2 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SAI2CLKSOURCE_PLL: SAI2 clock = PLL
 - RCC_SAI2CLKSOURCE_PLL2: SAI2 clock = PLL2
 - RCC_SAI2CLKSOURCE_PLL3: SAI2 clock = PLL3
 - RCC_SAI2CLKSOURCE_CLKP: SAI2 clock = CLKp
 - RCC_SAI2CLKSOURCE_PIN: SAI2 clock = External Clock

__HAL_RCC_SAI3_CONFIG

Description:

- Macro to Configure the SAI3 clock source.

Parameters:

- __RCC_SAI3CLKSource__: defines the SAI3 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SAI3CLKSOURCE_PLL: SAI3 clock = PLL
 - RCC_SAI3CLKSOURCE_PLL2: SAI3 clock = PLL2
 - RCC_SAI3CLKSOURCE_PLL3: SAI3 clock = PLL3
 - RCC_SAI3CLKSOURCE_CLKP: SAI3 clock = CLKp
 - RCC_SAI3CLKSOURCE_PIN: SAI3 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI3_SOURCE

Description:

- Macro to get the SAI3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SAI3CLKSOURCE_PLL: SAI3 clock = PLL

- RCC_SAI3CLKSOURCE_PLL2: SAI3 clock = PLL2
- RCC_SAI3CLKSOURCE_PLL3: SAI3 clock = PLL3
- RCC_SAI3CLKSOURCE_CLKP: SAI3 clock = CLKP
- RCC_SAI3CLKSOURCE_PIN: SAI3 clock = External Clock

__HAL_RCC_SAI4A_CONFIG

Description:

- Macro to Configure the SAI4A clock source.

Parameters:

- RCC_SAI4ACLKSource: defines the SAI4A clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SAI4ACLKSOURCE_PLL: SAI4A clock = PLL
 - RCC_SAI4ACLKSOURCE_PLL2: SAI4A clock = PLL2
 - RCC_SAI4ACLKSOURCE_PLL3: SAI4A clock = PLL3
 - RCC_SAI4ACLKSOURCE_CLKP: SAI4A clock = CLKP
 - RCC_SAI4ACLKSOURCE_PIN: SAI4A clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI4A_SOURCE

Description:

- Macro to get the SAI4A clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SAI4ACLKSOURCE_PLL: SAI4B clock = PLL
 - RCC_SAI4ACLKSOURCE_PLL2: SAI4B clock = PLL2
 - RCC_SAI4ACLKSOURCE_PLL3: SAI4B clock = PLL3
 - RCC_SAI4ACLKSOURCE_CLKP: SAI4B clock = CLKp
 - RCC_SAI4ACLKSOURCE_PIN: SAI4B clock = External Clock

__HAL_RCC_SAI4B_CONFIG

Description:

- Macro to Configure the SAI4B clock source.

Parameters:

- RCC_SAI4BCLKSource: defines the SAI4B clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SAI4BCLKSOURCE_PLL: SAI4B clock = PLL
 - RCC_SAI4BCLKSOURCE_PLL2: SAI4B clock = PLL2
 - RCC_SAI4BCLKSOURCE_PLL3: SAI4B clock = PLL3
 - RCC_SAI4BCLKSOURCE_CLKP: SAI4B clock = CLKp
 - RCC_SAI4BCLKSOURCE_PIN: SAI4B clock = External Clock

Return value:

- None

__HAL_RCC_GET_SAI4B_SOURCE

Description:

- Macro to get the SAI4B clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SAI4BCLKSOURCE_PLL: SAI4B clock = PLL
 - RCC_SAI4BCLKSOURCE_PLL2: SAI4B clock = PLL2
 - RCC_SAI4BCLKSOURCE_PLL3: SAI4B clock = PLL3
 - RCC_SAI4BCLKSOURCE_CLKP: SAI4B clock = CLKP
 - RCC_SAI4BCLKSOURCE_PIN: SAI4B clock = External Clock

__HAL_RCC_I2C123_CONFIG

Description:

- macro to configure the I2C1/2/3 clock (I2C123CLK).

Parameters:

- __I2C123CLKSource__: specifies the I2C1/2/3 clock source. This parameter can be one of the following values:
 - RCC_I2C123CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C1/2/3 clock
 - RCC_I2C123CLKSOURCE_PLL3: PLL3 selected as I2C1/2/3 clock
 - RCC_I2C123CLKSOURCE_HSI: HSI selected as I2C1/2/3 clock
 - RCC_I2C123CLKSOURCE_CSI: CSI selected as I2C1/2/3 clock

__HAL_RCC_GET_I2C123_SOURCE

Description:

- macro to get the I2C1/2/3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_I2C123CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C1/2/3 clock
 - RCC_I2C123CLKSOURCE_PLL3: PLL3 selected as I2C1/2/3 clock
 - RCC_I2C123CLKSOURCE_HSI: HSI selected as I2C1/2/3 clock
 - RCC_I2C123CLKSOURCE_CSI: CSI selected as I2C1/2/3 clock

__HAL_RCC_I2C1_CONFIG

Description:

- macro to configure the I2C1 clock (I2C1CLK).

Parameters:

- __I2C1CLKSource__: specifies the I2C1 clock source. This parameter can be one of the following values:
 - RCC_I2C1CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C1 clock
 - RCC_I2C1CLKSOURCE_PLL3: PLL3 selected as I2C1 clock
 - RCC_I2C1CLKSOURCE_HSI: HSI selected as I2C1 clock
 - RCC_I2C1CLKSOURCE_CSI: CSI selected as I2C1 clock

__HAL_RCC_GET_I2C1_SOURCE

Description:

- macro to get the I2C1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_I2C1CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C1 clock
 - RCC_I2C1CLKSOURCE_PLL3: PLL3 selected as I2C1 clock
 - RCC_I2C1CLKSOURCE_HSI: HSI selected as I2C1 clock

- RCC_I2C1CLKSOURCE_CSI: CSI selected as I2C1 clock

__HAL_RCC_I2C2_CONFIG

Description:

- macro to configure the I2C2 clock (I2C2CLK).

Parameters:

- __I2C2CLKSource__: specifies the I2C2 clock source. This parameter can be one of the following values:
 - RCC_I2C2CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C2 clock
 - RCC_I2C2CLKSOURCE_PLL3: PLL3 selected as I2C2 clock
 - RCC_I2C2CLKSOURCE_HSI: HSI selected as I2C2 clock
 - RCC_I2C2CLKSOURCE_CSI: CSI selected as I2C2 clock

__HAL_RCC_GET_I2C2_SOURCE

Description:

- macro to get the I2C2 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_I2C2CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C2 clock
 - RCC_I2C2CLKSOURCE_PLL3: PLL3 selected as I2C2 clock
 - RCC_I2C2CLKSOURCE_HSI: HSI selected as I2C2 clock
 - RCC_I2C2CLKSOURCE_CSI: CSI selected as I2C2 clock

__HAL_RCC_I2C3_CONFIG

Description:

- macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- __I2C3CLKSource__: specifies the I2C3 clock source. This parameter can be one of the following values:
 - RCC_I2C3CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C3 clock
 - RCC_I2C3CLKSOURCE_PLL3: PLL3 selected as I2C3 clock
 - RCC_I2C3CLKSOURCE_HSI: HSI selected as I2C3 clock
 - RCC_I2C3CLKSOURCE_CSI: CSI selected as I2C3 clock

__HAL_RCC_GET_I2C3_SOURCE

Description:

- macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_I2C3CLKSOURCE_D2PCLK1: D2PCLK1 selected as I2C3 clock
 - RCC_I2C3CLKSOURCE_PLL3: PLL3 selected as I2C3 clock
 - RCC_I2C3CLKSOURCE_HSI: HSI selected as I2C3 clock
 - RCC_I2C3CLKSOURCE_CSI: CSI selected as I2C3 clock

__HAL_RCC_I2C4_CONFIG

Description:

- macro to configure the I2C4 clock (I2C4CLK).

Parameters:

- __I2C4CLKSource__: specifies the I2C4 clock source. This parameter can be one of the following values:
 - RCC_I2C4CLKSOURCE_D3PCLK1: D3PCLK1 selected as I2C4 clock

- RCC_I2C4CLKSOURCE_PLL3: PLL3 selected as I2C4 clock
- RCC_I2C4CLKSOURCE_HSI: HSI selected as I2C4 clock
- RCC_I2C4CLKSOURCE_CSI: CSI selected as I2C4 clock

__HAL_RCC_GET_I2C4_SOURCE

Description:

- macro to get the I2C4 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_I2C4CLKSOURCE_D3PCLK1: D3PCLK1 selected as I2C4 clock
 - RCC_I2C4CLKSOURCE_PLL3: PLL3 selected as I2C4 clock
 - RCC_I2C4CLKSOURCE_HSI: HSI selected as I2C4 clock
 - RCC_I2C4CLKSOURCE_CSI: CSI selected as I2C4 clock

__HAL_RCC_USART16_CONFIG

Description:

- macro to configure the USART1/6 clock (USART16CLK).

Parameters:

- USART16CLKSource: specifies the USART1/6 clock source. This parameter can be one of the following values:
 - RCC_USART16CLKSOURCE_D2PCLK2: APB2 Clock selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_PLL2: PLL2_Q Clock selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_PLL3: PLL3_Q Clock selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_HSI: HSI selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_CSI: CSI Clock selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_LSE: LSE selected as USART1/6 clock

__HAL_RCC_GET_USART16_SOURCE

Description:

- macro to get the USART1/6 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USART16CLKSOURCE_D2PCLK2: APB2 Clock selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_PLL2: PLL2_Q Clock selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_PLL3: PLL3_Q Clock selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_HSI: HSI selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_CSI: CSI Clock selected as USART1/6 clock
 - RCC_USART16CLKSOURCE_LSE: LSE selected as USART1/6 clock

__HAL_RCC_USART234578_CONFIG

Description:

- macro to configure the USART234578 clock (USART234578CLK).

Parameters:

- USART234578CLKSource: specifies the USART2/3/4/5/7/8 clock source. This parameter can be one of the following values:
 - RCC_USART234578CLKSOURCE_D2PCLK1: APB1 Clock selected as USART2/3/4/5/7/8 clock
 - RCC_USART234578CLKSOURCE_PLL2: PLL2_Q Clock selected as USART2/3/4/5/7/8 clock
 - RCC_USART234578CLKSOURCE_PLL3: PLL3_Q Clock selected as USART2/3/4/5/7/8 clock

- RCC_USART234578CLKSOURCE_HSI: HSI selected as USART2/3/4/5/7/8 clock
- RCC_USART234578CLKSOURCE_CSI: CSI Clock selected as USART2/3/4/5/7/8 clock
- RCC_USART234578CLKSOURCE_LSE: LSE selected as USART2/3/4/5/7/8 clock

__HAL_RCC_GET_USART234578_SOURCE

Description:

- macro to get the USART2/3/4/5/7/8 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USART234578CLKSOURCE_D2PCLK1: APB1 Clock selected as USART2/3/4/5/7/8 clock
 - RCC_USART234578CLKSOURCE_PLL2: PLL2_Q Clock selected as USART2/3/4/5/7/8 clock
 - RCC_USART234578CLKSOURCE_PLL3: PLL3_Q Clock selected as USART2/3/4/5/7/8 clock
 - RCC_USART234578CLKSOURCE_HSI: HSI selected as USART2/3/4/5/7/8 clock
 - RCC_USART234578CLKSOURCE_CSI: CSI Clock selected as USART2/3/4/5/7/8 clock
 - RCC_USART234578CLKSOURCE_LSE: LSE selected as USART2/3/4/5/7/8 clock

__HAL_RCC_USART1_CONFIG

Description:

- macro to configure the USART1 clock (USART1CLK).

Parameters:

- _USART1CLKSource: specifies the USART1 clock source. This parameter can be one of the following values:
 - RCC_USART1CLKSOURCE_D2PCLK2: APB2 Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_PLL2: PLL2_Q Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_PLL3: PLL3_Q Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_HSI: HSI selected as USART1 clock
 - RCC_USART1CLKSOURCE_CSI: CSI Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_LSE: LSE selected as USART1 clock

__HAL_RCC_GET_USART1_SOURCE

Description:

- macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USART1CLKSOURCE_D2PCLK2: APB2 Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_PLL2: PLL2_Q Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_PLL3: PLL3_Q Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_HSI: HSI selected as USART1 clock
 - RCC_USART1CLKSOURCE_CSI: CSI Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_LSE: LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- macro to configure the USART2 clock (USART2CLK).

Parameters:

- _USART2CLKSource: specifies the USART2 clock source. This parameter can be one of the following values:
 - RCC_USART2CLKSOURCE_D2PCLK1: APB1 Clock selected as USART2 clock

- RCC_USART2CLKSOURCE_PLL2: PLL2_Q Clock selected as USART2 clock
- RCC_USART2CLKSOURCE_PLL3: PLL3_Q Clock selected as USART2 clock
- RCC_USART2CLKSOURCE_HSI: HSI selected as USART2 clock
- RCC_USART2CLKSOURCE_CSI: CSI Clock selected as USART2 clock
- RCC_USART2CLKSOURCE_LSE: LSE selected as USART2 clock

[__HAL_RCC_GET_USART2_SOURCE](#)

Description:

- macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USART2CLKSOURCE_D2PCLK1: APB1 Clock selected as USART2 clock
 - RCC_USART2CLKSOURCE_PLL2: PLL2_Q Clock selected as USART2 clock
 - RCC_USART2CLKSOURCE_PLL3: PLL3_Q Clock selected as USART2 clock
 - RCC_USART2CLKSOURCE_HSI: HSI selected as USART2 clock
 - RCC_USART2CLKSOURCE_CSI: CSI Clock selected as USART2 clock
 - RCC_USART2CLKSOURCE_LSE: LSE selected as USART2 clock

[__HAL_RCC_USART3_CONFIG](#)

Description:

- macro to configure the USART3 clock (USART3CLK).

Parameters:

- __USART3CLKSource__: specifies the USART3 clock source. This parameter can be one of the following values:
 - RCC_USART3CLKSOURCE_D2PCLK1: APB1 Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_PLL2: PLL2_Q Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_PLL3: PLL3_Q Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_HSI: HSI selected as USART3 clock
 - RCC_USART3CLKSOURCE_CSI: CSI Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_LSE: LSE selected as USART3 clock

[__HAL_RCC_GET_USART3_SOURCE](#)

Description:

- macro to get the USART3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USART2CLKSOURCE_D2PCLK1: APB1 Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_PLL2: PLL2_Q Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_PLL3: PLL3_Q Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_HSI: HSI selected as USART3 clock
 - RCC_USART3CLKSOURCE_CSI: CSI Clock selected as USART3 clock
 - RCC_USART3CLKSOURCE_LSE: LSE selected as USART3 clock

[__HAL_RCC_UART4_CONFIG](#)

Description:

- macro to configure the UART4 clock (UART4CLK).

Parameters:

- UART4CLKSource: specifies the UART4 clock source. This parameter can be one of the following values:
 - RCC_UART4CLKSOURCE_D2PCLK1: APB1 Clock selected as UART4 clock
 - RCC_UART4CLKSOURCE_PLL2: PLL2_Q Clock selected as UART4 clock
 - RCC_UART4CLKSOURCE_PLL3: PLL3_Q Clock selected as UART4 clock
 - RCC_UART4CLKSOURCE_HSI: HSI selected as UART4 clock
 - RCC_UART4CLKSOURCE_CSI: CSI Clock selected as UART4 clock
 - RCC_UART4CLKSOURCE_LSE: LSE selected as UART4 clock

__HAL_RCC_GET_UART4_SOURCE

Description:

- macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_UART4CLKSOURCE_D2PCLK1: APB1 Clock selected as UART4 clock
 - RCC_UART4CLKSOURCE_PLL2: PLL2_Q Clock selected as UART4 clock
 - RCC_UART4CLKSOURCE_PLL3: PLL3_Q Clock selected as UART4 clock
 - RCC_UART4CLKSOURCE_HSI: HSI selected as UART4 clock
 - RCC_UART4CLKSOURCE_CSI: CSI Clock selected as UART4 clock
 - RCC_UART4CLKSOURCE_LSE: LSE selected as UART4 clock

__HAL_RCC_UART5_CONFIG

Description:

- macro to configure the UART5 clock (UART5CLK).

Parameters:

- UART5CLKSource: specifies the UART5 clock source. This parameter can be one of the following values:
 - RCC_UART5CLKSOURCE_D2PCLK1: APB1 Clock selected as UART5 clock
 - RCC_UART5CLKSOURCE_PLL2: PLL2_Q Clock selected as UART5 clock
 - RCC_UART5CLKSOURCE_PLL3: PLL3_Q Clock selected as UART5 clock
 - RCC_UART5CLKSOURCE_HSI: HSI selected as UART5 clock
 - RCC_UART5CLKSOURCE_CSI: CSI Clock selected as UART5 clock
 - RCC_UART5CLKSOURCE_LSE: LSE selected as UART5 clock

__HAL_RCC_GET_UART5_SOURCE

Description:

- macro to get the UART5 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_UART5CLKSOURCE_D2PCLK1: APB1 Clock selected as UART5 clock
 - RCC_UART5CLKSOURCE_PLL2: PLL2_Q Clock selected as UART5 clock
 - RCC_UART5CLKSOURCE_PLL3: PLL3_Q Clock selected as UART5 clock
 - RCC_UART5CLKSOURCE_HSI: HSI selected as UART5 clock
 - RCC_UART5CLKSOURCE_CSI: CSI Clock selected as UART5 clock
 - RCC_UART5CLKSOURCE_LSE: LSE selected as UART5 clock

__HAL_RCC_USART6_CONFIG

Description:

- macro to configure the USART6 clock (USART6CLK).

Parameters:

- __USART6CLKSource__: specifies the USART6 clock source. This parameter can be one of the following values:
 - RCC_USART6CLKSOURCE_D2PCLK2: APB2 Clock selected as USART6 clock
 - RCC_USART6CLKSOURCE_PLL2: PLL2_Q Clock selected as USART6 clock
 - RCC_USART6CLKSOURCE_PLL3: PLL3_Q Clock selected as USART6 clock
 - RCC_USART6CLKSOURCE_HSI: HSI selected as USART6 clock
 - RCC_USART6CLKSOURCE_CSI: CSI Clock selected as USART6 clock
 - RCC_USART6CLKSOURCE_LSE: LSE selected as USART6 clock

[__HAL_RCC_GET_USART6_SOURCE](#)**Description:**

- macro to get the USART6 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USART6CLKSOURCE_D2PCLK2: APB2 Clock selected as USART6 clock
 - RCC_USART6CLKSOURCE_PLL2: PLL2_Q Clock selected as USART6 clock
 - RCC_USART6CLKSOURCE_PLL3: PLL3_Q Clock selected as USART6 clock
 - RCC_USART6CLKSOURCE_HSI: HSI selected as USART6 clock
 - RCC_USART6CLKSOURCE_CSI: CSI Clock selected as USART6 clock
 - RCC_USART6CLKSOURCE_LSE: LSE selected as USART6 clock

[__HAL_RCC_UART7_CONFIG](#)**Description:**

- macro to configure the UART5 clock (UART7CLK).

Parameters:

- __UART7CLKSource__: specifies the UART7 clock source. This parameter can be one of the following values:
 - RCC_UART7CLKSOURCE_D2PCLK1: APB1 Clock selected as UART7 clock
 - RCC_UART7CLKSOURCE_PLL2: PLL2_Q Clock selected as UART7 clock
 - RCC_UART7CLKSOURCE_PLL3: PLL3_Q Clock selected as UART7 clock
 - RCC_UART7CLKSOURCE_HSI: HSI selected as UART7 clock
 - RCC_UART7CLKSOURCE_CSI: CSI Clock selected as UART7 clock
 - RCC_UART7CLKSOURCE_LSE: LSE selected as UART7 clock

[__HAL_RCC_GET_UART7_SOURCE](#)**Description:**

- macro to get the UART7 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_UART7CLKSOURCE_D2PCLK1: APB1 Clock selected as UART7 clock
 - RCC_UART7CLKSOURCE_PLL2: PLL2_Q Clock selected as UART7 clock
 - RCC_UART7CLKSOURCE_PLL3: PLL3_Q Clock selected as UART7 clock
 - RCC_UART7CLKSOURCE_HSI: HSI selected as UART7 clock
 - RCC_UART7CLKSOURCE_CSI: CSI Clock selected as UART7 clock
 - RCC_UART7CLKSOURCE_LSE: LSE selected as UART7 clock

[__HAL_RCC_UART8_CONFIG](#)**Description:**

- macro to configure the UART8 clock (UART8CLK).

Parameters:

- `_UART8CLKSource_`: specifies the UART8 clock source. This parameter can be one of the following values:
 - `RCC_UART8CLKSOURCE_D2PCLK1`: APB1 Clock selected as UART8 clock
 - `RCC_UART8CLKSOURCE_PLL2`: PLL2_Q Clock selected as UART8 clock
 - `RCC_UART8CLKSOURCE_PLL3`: PLL3_Q Clock selected as UART8 clock
 - `RCC_UART8CLKSOURCE_HSI`: HSI selected as UART8 clock
 - `RCC_UART8CLKSOURCE_CSI`: CSI Clock selected as UART8 clock
 - `RCC_UART8CLKSOURCE_LSE`: LSE selected as UART8 clock

[`__HAL_RCC_GET_UART8_SOURCE`](#)**Description:**

- macro to get the UART8 clock source.

Return value:

- The: clock source can be one of the following values:
 - `RCC_UART8CLKSOURCE_D2PCLK1`: APB1 Clock selected as UART8 clock
 - `RCC_UART8CLKSOURCE_PLL2`: PLL2_Q Clock selected as UART8 clock
 - `RCC_UART8CLKSOURCE_PLL3`: PLL3_Q Clock selected as UART8 clock
 - `RCC_UART8CLKSOURCE_HSI`: HSI selected as UART8 clock
 - `RCC_UART8CLKSOURCE_CSI`: CSI Clock selected as UART8 clock
 - `RCC_UART8CLKSOURCE_LSE`: LSE selected as UART8 clock

[`__HAL_RCC_LPUART1_CONFIG`](#)**Description:**

- macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:

- `_LPUART1CLKSource_`: specifies the LPUART1 clock source. This parameter can be one of the following values:
 - `RCC_LPUART1CLKSOURCE_D3PCLK1`: APB4 Clock selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_PLL2`: PLL2_Q Clock selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_PLL3`: PLL3_Q Clock selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_HSI`: HSI selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_CSI`: CSI Clock selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_LSE`: LSE selected as LPUART1 clock

[`__HAL_RCC_GET_LPUART1_SOURCE`](#)**Description:**

- macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
 - `RCC_LPUART1CLKSOURCE_D3PCLK1`: APB4 Clock selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_PLL2`: PLL2_Q Clock selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_PLL3`: PLL3_Q Clock selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_HSI`: HSI selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_CSI`: CSI Clock selected as LPUART1 clock
 - `RCC_LPUART1CLKSOURCE_LSE`: LSE selected as LPUART1 clock

__HAL_RCC_LPTIM1_CONFIG

Description:

- macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM1CLKSOURCE_D2PCLK1: APB1 Clock selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_LSE: LSE selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_LSI: LSI Clock selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_CLKP: CLKP selected as LPTIM1 clock

__HAL_RCC_GET_LPTIM1_SOURCE

Description:

- macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM1CLKSOURCE_D2PCLK1: APB1 Clock selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_LSE: LSE selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_LSI: LSI Clock selected as LPTIM1 clock
 - RCC_LPTIM1CLKSOURCE_CLKP: CLKP selected as LPTIM1 clock

__HAL_RCC_LPTIM2_CONFIG

Description:

- macro to get the LPTIM2 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM2CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_LSE: LSE selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_LSI: LSI Clock selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_CLKP: CLKP selected as LPTIM2 clock

__HAL_RCC_GET_LPTIM2_SOURCE

Description:

- macro to get the LPTIM2 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM2CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_LSE: LSE selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_LSI: LSI Clock selected as LPTIM2 clock
 - RCC_LPTIM2CLKSOURCE_CLKP: CLKP selected as LPTIM2 clock

__HAL_RCC_LPTIM345_CONFIG

Description:

- macro to get the LPTIM3/4/5 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM345CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_LSE: LSE selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_LSI: LSI Clock selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_CLKP: CLKP selected as LPTIM3/4/5 clock

__HAL_RCC_GET_LPTIM345_SOURCE

Description:

- macro to get the LPTIM3/4/5 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM345CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_LSE: LSE selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_LSI: LSI Clock selected as LPTIM3/4/5 clock
 - RCC_LPTIM345CLKSOURCE_CLKP: CLKP selected as LPTIM3/4/5 clock

__HAL_RCC_LPTIM3_CONFIG

Description:

- macro to get the LPTIM3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM3CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_LSE: LSE selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_LSI: LSI Clock selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_CLKP: CLKP selected as LPTIM3 clock

__HAL_RCC_GET_LPTIM3_SOURCE

Description:

- macro to get the LPTIM3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM3CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_LSE: LSE selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_LSI: LSI Clock selected as LPTIM3 clock
 - RCC_LPTIM3CLKSOURCE_CLKP: CLKP selected as LPTIM3 clock

__HAL_RCC_LPTIM4_CONFIG

Description:

- macro to get the LPTIM4 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM4CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_LSE: LSE selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_LSI: LSI Clock selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_CLKP: CLKP selected as LPTIM4 clock

__HAL_RCC_GET_LPTIM4_SOURCE

Description:

- macro to get the LPTIM4 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM4CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_LSE: LSE selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_LSI: LSI Clock selected as LPTIM4 clock
 - RCC_LPTIM4CLKSOURCE_CLKP: CLKP selected as LPTIM4 clock

__HAL_RCC_LPTIM5_CONFIG

Description:

- macro to configure the LPTIM5 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM5CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_LSE: LSE selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_LSI: LSI Clock selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_CLKP: CLKP selected as LPTIM5 clock

__HAL_RCC_GET_LPTIM5_SOURCE

Description:

- macro to get the LPTIM5 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_LPTIM5CLKSOURCE_D3PCLK1: APB4 Clock selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_PLL2: PLL2_P Clock selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_PLL3: PLL3_R Clock selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_LSE: LSE selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_LSI: LSI Clock selected as LPTIM5 clock
 - RCC_LPTIM5CLKSOURCE_CLKP: CLKP selected as LPTIM5 clock

__HAL_RCC_QSPI_CONFIG

Description:

- macro to configure the QSPI clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_RCC_QSPICLKSOURCE_D1HCLK: Domain1 HCLK Clock selected as QSPI clock
 - RCC_RCC_QSPICLKSOURCE_PLL : PLL1_Q Clock selected as QSPI clock
 - RCC_RCC_QSPICLKSOURCE_PLL2 : PLL2_R Clock selected as QSPI clock
 - RCC_RCC_QSPICLKSOURCE_CLKP CLKP selected as QSPI clock

__HAL_RCC_GET_QSPI_SOURCE

Description:

- macro to get the QSPI clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_RCC_QSPICLKSOURCE_D1HCLK: Domain1 HCLK Clock selected as QSPI clock
 - RCC_RCC_QSPICLKSOURCE_PLL : PLL1_Q Clock selected as QSPI clock
 - RCC_RCC_QSPICLKSOURCE_PLL2 : PLL2_R Clock selected as QSPI clock
 - RCC_RCC_QSPICLKSOURCE_CLKP CLKP selected as QSPI clock

__HAL_RCC_FMC_CONFIG

Description:

- macro to configure the FMC clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_RCC_FMCCCLKSOURCE_D1HCLK: Domain1 HCLK Clock selected as FMC clock
 - RCC_RCC_FMCCCLKSOURCE_PLL : PLL1_Q Clock selected as FMC clock
 - RCC_RCC_FMCCCLKSOURCE_PLL2 : PLL2_R Clock selected as FMC clock
 - RCC_RCC_FMCCCLKSOURCE_CLKP CLKP selected as FMC clock

__HAL_RCC_GET_FMC_SOURCE

Description:

- macro to get the FMC clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_RCC_FMCCCLKSOURCE_D1HCLK: Domain1 HCLK Clock selected as FMC clock
 - RCC_RCC_FMCCCLKSOURCE_PLL : PLL1_Q Clock selected as FMC clock
 - RCC_RCC_FMCCCLKSOURCE_PLL2 : PLL2_R Clock selected as FMC clock
 - RCC_RCC_FMCCCLKSOURCE_CLKP CLKP selected as FMC clock

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock (USBCLK).

Parameters:

- __USBCLKSource__: specifies the USB clock source. This parameter can be one of the following values:
 - RCC_USBCLKSOURCE_PLL: PLL1Q selected as USB clock
 - RCC_USBCLKSOURCE_PLL3: PLL3Q Clock selected as USB clock

- RCC_USBCLKSOURCE_HSI48: RC48 MHZ Clock selected as USB clock

__HAL_RCC_GET_USB_SOURCE

Description:

- Macro to get the USB clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USBCLKSOURCE_PLL: PLL1Q selected as USB clock
 - RCC_USBCLKSOURCE_PLL3: PLL3Q Clock selected as USB clock
 - RCC_USBCLKSOURCE_HSI48: RC48 MHZ Clock selected as USB clock

__HAL_RCC_ADC_CONFIG

Description:

- Macro to configure the ADC clock.

Parameters:

- ADCCLKSource: specifies the ADC digital interface clock source. This parameter can be one of the following values:
 - RCC_ADCCLKSOURCE_PLL2: PLL2_P Clock selected as ADC clock
 - RCC_ADCCLKSOURCE_PLL3: PLL3_R Clock selected as ADC clock
 - RCC_ADCCLKSOURCE_CLKP: CLKP Clock selected as ADC clock

__HAL_RCC_GET_ADC_SOURCE

Description:

- Macro to get the ADC clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_ADCCLKSOURCE_PLL2: PLL2_P Clock selected as ADC clock
 - RCC_ADCCLKSOURCE_PLL3: PLL3_R Clock selected as ADC clock
 - RCC_ADCCLKSOURCE_CLKP: CLKP Clock selected as ADC clock

__HAL_RCC_SWPMI1_CONFIG

Description:

- Macro to configure the SWPMI1 clock.

Parameters:

- SWPMI1CLKSource: specifies the SWPMI1 clock source. This parameter can be one of the following values:
 - RCC_SWPMI1CLKSOURCE_D2PCLK1: D2PCLK1 Clock selected as SWPMI1 clock
 - RCC_SWPMI1CLKSOURCE_HSI: HSI Clock selected as SWPMI1 clock

__HAL_RCC_GET_SWPMI1_SOURCE

Description:

- Macro to get the SWPMI1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SWPMI1CLKSOURCE_D2PCLK1: D2PCLK1 Clock selected as SWPMI1 clock
 - RCC_SWPMI1CLKSOURCE_HSI: HSI Clock selected as SWPMI1 clock

__HAL_RCC_DFSDM1_CONFIG

Description:

- Macro to configure the DFSDM1 clock.

Parameters:

- __DFSDM1CLKSource__: specifies the DFSDM1 clock source. This parameter can be one of the following values:
 - RCC_DFSDM1CLKSOURCE_D2PCLK: D2PCLK Clock selected as DFSDM1 clock
 - RCC_DFSDM1CLKSOURCE_SYS: System Clock selected as DFSDM1 clock

__HAL_RCC_GET_DFSDM1_SOURCE

Description:

- Macro to get the DFSDM1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_DFSDM1CLKSOURCE_D2PCLK: D2PCLK Clock selected as DFSDM1 clock
 - RCC_DFSDM1CLKSOURCE_SYS: System Clock selected as DFSDM1 clock

__HAL_RCC_CEC_CONFIG

Description:

- macro to configure the CEC clock (CECCLK).

Parameters:

- __CECCLKSource__: specifies the CEC clock source. This parameter can be one of the following values:
 - RCC_CECCLKSOURCE_LSE: LSE selected as CEC clock
 - RCC_CECCLKSOURCE_LSI: LSI selected as CEC clock
 - RCC_CECCLKSOURCE_CSI: CSI Clock selected as CEC clock

__HAL_RCC_GET_CEC_SOURCE

Description:

- macro to get the CEC clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_CECCLKSOURCE_LSE: LSE selected as CEC clock
 - RCC_CECCLKSOURCE_LSI: LSI selected as CEC clock
 - RCC_CECCLKSOURCE_CSI: CSI Clock selected as CEC clock

__HAL_RCC_CLKP_CONFIG

Description:

- Macro to configure the CLKP : Oscillator clock for peripheral.

Parameters:

- __CLKPSource__: specifies Oscillator clock for peripheral This parameter can be one of the following values:
 - RCC_CLKPSOURCE_HSI: HSI selected Oscillator clock for peripheral
 - RCC_CLKPSOURCE_CSI: CSI selected Oscillator clock for peripheral
 - RCC_CLKPSOURCE_HSE: HSE selected Oscillator clock for peripheral

__HAL_RCC_GET_CLKP_SOURCE

Description:

- Macro to get the Oscillator clock for peripheral source.

Return value:

- The: clock source can be one of the following values:
 - RCC_CLKPSOURCE_HSI: HSI selected Oscillator clock for peripheral
 - RCC_CLKPSOURCE_CSI: CSI selected Oscillator clock for peripheral
 - RCC_CLKPSOURCE_HSE: HSE selected Oscillator clock for peripheral

[__HAL_RCC_FDCAN_CONFIG](#)

Description:

- Macro to configure the FDCAN clock.

Parameters:

- __FDCANCLKSource: specifies clock source for FDCAN This parameter can be one of the following values:
 - RCC_FDCANCLKSOURCE_HSE: HSE selected as FDCAN clock
 - RCC_FDCANCLKSOURCE_PLL: PLL selected as FDCAN clock
 - RCC_FDCANCLKSOURCE_PLL2: PLL2 selected as FDCAN clock

[__HAL_RCC_GET_FDCAN_SOURCE](#)

Description:

- Macro to get the FDCAN clock.

Return value:

- The: clock source can be one of the following values:
 - RCC_FDCANCLKSOURCE_HSE: HSE selected as FDCAN clock
 - RCC_FDCANCLKSOURCE_PLL: PLL selected as FDCAN clock
 - RCC_FDCANCLKSOURCE_PLL2: PLL2 selected as FDCAN clock

[__HAL_RCC_SPI123_CONFIG](#)

Description:

- Macro to Configure the SPI1/2/3 clock source.

Parameters:

- __RCC_SPI123CLKSource: defines the SPI1/2/3 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SPI123CLKSOURCE_PLL: SPI1/2/3 clock = PLL
 - RCC_SPI123CLKSOURCE_PLL2: SPI1/2/3 clock = PLL2
 - RCC_SPI123CLKSOURCE_PLL3: SPI1/2/3 clock = PLL3
 - RCC_SPI123CLKSOURCE_CLKP: SPI1/2/3 clock = CLKP
 - RCC_SPI123CLKSOURCE_PIN: SPI1/2/3 clock = External Clock

Return value:

- None

[__HAL_RCC_GET_SPI123_SOURCE](#)

Description:

- Macro to get the SPI1/2/3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SPI123CLKSOURCE_PLL: SPI1/2/3 clock = PLL
 - RCC_SPI123CLKSOURCE_PLL2: SPI1/2/3 clock = PLL2

- RCC_SPI123CLKSOURCE_PLL3: SPI1/2/3 clock = PLL3
- RCC_SPI123CLKSOURCE_CLKP: SPI1/2/3 clock = CLKp
- RCC_SPI123CLKSOURCE_PIN: SPI1/2/3 clock = External Clock

__HAL_RCC_SPI1_CONFIG

Description:

- Macro to Configure the SPI1 clock source.

Parameters:

- RCC_SPI1CLKSource: defines the SPI1 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SPI1CLKSOURCE_PLL: SPI1 clock = PLL
 - RCC_SPI1CLKSOURCE_PLL2: SPI1 clock = PLL2
 - RCC_SPI1CLKSOURCE_PLL3: SPI1 clock = PLL3
 - RCC_SPI1CLKSOURCE_CLKP: SPI1 clock = CLKp
 - RCC_SPI1CLKSOURCE_PIN: SPI1 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SPI1_SOURCE

Description:

- Macro to get the SPI1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SPI1CLKSOURCE_PLL: SPI1 clock = PLL
 - RCC_SPI1CLKSOURCE_PLL2: SPI1 clock = PLL2
 - RCC_SPI1CLKSOURCE_PLL3: SPI1 clock = PLL3
 - RCC_SPI1CLKSOURCE_CLKP: SPI1 clock = CLKp
 - RCC_SPI1CLKSOURCE_PIN: SPI1 clock = External Clock

__HAL_RCC_SPI2_CONFIG

Description:

- Macro to Configure the SPI2 clock source.

Parameters:

- RCC_SPI2CLKSource: defines the SPI2 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SPI2CLKSOURCE_PLL: SPI2 clock = PLL
 - RCC_SPI2CLKSOURCE_PLL2: SPI2 clock = PLL2
 - RCC_SPI2CLKSOURCE_PLL3: SPI2 clock = PLL3
 - RCC_SPI2CLKSOURCE_CLKP: SPI2 clock = CLKp
 - RCC_SPI2CLKSOURCE_PIN: SPI2 clock = External Clock

Return value:

- None

__HAL_RCC_GET_SPI2_SOURCE

Description:

- Macro to get the SPI2 clock source.

Return value:

- The: clock source can be one of the following values:

- RCC_SPI2CLKSOURCE_PLL: SPI2 clock = PLL
- RCC_SPI2CLKSOURCE_PLL2: SPI2 clock = PLL2
- RCC_SPI2CLKSOURCE_PLL3: SPI2 clock = PLL3
- RCC_SPI2CLKSOURCE_CLKP: SPI2 clock = CLKp
- RCC_SPI2CLKSOURCE_PIN: SPI2 clock = External Clock

[__HAL_RCC_SPI3_CONFIG](#)

Description:

- Macro to Configure the SPI3 clock source.

Parameters:

- RCC_SPI3CLKSource: defines the SPI3 clock source. This clock is derived from system PLL, PLL2, PLL3, OSC or external clock (through a dedicated PIN) This parameter can be one of the following values:
 - RCC_SPI3CLKSOURCE_PLL: SPI3 clock = PLL
 - RCC_SPI3CLKSOURCE_PLL2: SPI3 clock = PLL2
 - RCC_SPI3CLKSOURCE_PLL3: SPI3 clock = PLL3
 - RCC_SPI3CLKSOURCE_CLKP: SPI3 clock = CLKp
 - RCC_SPI3CLKSOURCE_PIN: SPI3 clock = External Clock

Return value:

- None

[__HAL_RCC_GET_SPI3_SOURCE](#)

Description:

- Macro to get the SPI3 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SPI3CLKSOURCE_PLL: SPI3 clock = PLL
 - RCC_SPI3CLKSOURCE_PLL2: SPI3 clock = PLL2
 - RCC_SPI3CLKSOURCE_PLL3: SPI3 clock = PLL3
 - RCC_SPI3CLKSOURCE_CLKP: SPI3 clock = CLKp
 - RCC_SPI3CLKSOURCE_PIN: SPI3 clock = External Clock

[__HAL_RCC_SPI45_CONFIG](#)

Description:

- Macro to Configure the SPI4/5 clock source.

Parameters:

- RCC_SPI45CLKSource: defines the SPI4/5 clock source. This clock is derived from system PCLK, PLL2, PLL3, OSC This parameter can be one of the following values:
 - RCC_SPI45CLKSOURCE_D2PCLK1:SPI4/5 clock = D2PCLK1
 - RCC_SPI45CLKSOURCE_PLL2: SPI4/5 clock = PLL2
 - RCC_SPI45CLKSOURCE_PLL3: SPI4/5 clock = PLL3
 - RCC_SPI45CLKSOURCE_HSI: SPI4/5 clock = HSI
 - RCC_SPI45CLKSOURCE_CSI: SPI4/5 clock = CSI
 - RCC_SPI45CLKSOURCE_HSE: SPI4/5 clock = HSE

Return value:

- None

[__HAL_RCC_GET_SPI45_SOURCE](#)

Description:

- Macro to get the SPI4/5 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SPI4CLKSOURCE_D2PCLK1: SPI4/5 clock = D2PCLK1
 - RCC_SPI4CLKSOURCE_PLL2: SPI4/5 clock = PLL2
 - RCC_SPI4CLKSOURCE_PLL3: SPI4/5 clock = PLL3
 - RCC_SPI4CLKSOURCE_HSI: SPI4/5 clock = HSI
 - RCC_SPI4CLKSOURCE_CSI: SPI4/5 clock = CSI
 - RCC_SPI4CLKSOURCE_HSE: SPI4/5 clock = HSE

__HAL_RCC_SPI4_CONFIG

Description:

- Macro to Configure the SPI4 clock source.

Parameters:

- RCC_SPI4CLKSource: defines the SPI4 clock source. This clock is derived from system PCLK, PLL2, PLL3, OSC This parameter can be one of the following values:
 - RCC_SPI4CLKSOURCE_D2PCLK1: SPI4 clock = D2PCLK1
 - RCC_SPI4CLKSOURCE_PLL2: SPI4 clock = PLL2
 - RCC_SPI4CLKSOURCE_PLL3: SPI4 clock = PLL3
 - RCC_SPI4CLKSOURCE_HSI: SPI4 clock = HSI
 - RCC_SPI4CLKSOURCE_CSI: SPI4 clock = CSI
 - RCC_SPI4CLKSOURCE_HSE: SPI4 clock = HSE

Return value:

- None

__HAL_RCC_GET_SPI4_SOURCE

Description:

- Macro to get the SPI4 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SPI4CLKSOURCE_D2PCLK1: SPI4 clock = D2PCLK1
 - RCC_SPI4CLKSOURCE_PLL2: SPI4 clock = PLL2
 - RCC_SPI4CLKSOURCE_PLL3: SPI4 clock = PLL3
 - RCC_SPI4CLKSOURCE_HSI: SPI4 clock = HSI
 - RCC_SPI4CLKSOURCE_CSI: SPI4 clock = CSI
 - RCC_SPI4CLKSOURCE_HSE: SPI4 clock = HSE

__HAL_RCC_SPI5_CONFIG

Description:

- Macro to Configure the SPI5 clock source.

Parameters:

- RCC_SPI5CLKSource: defines the SPI5 clock source. This clock is derived from system PCLK, PLL2, PLL3, OSC This parameter can be one of the following values:
 - RCC_SPI5CLKSOURCE_D2PCLK1: SPI5 clock = D2PCLK1
 - RCC_SPI5CLKSOURCE_PLL2: SPI5 clock = PLL2
 - RCC_SPI5CLKSOURCE_PLL3: SPI5 clock = PLL3
 - RCC_SPI5CLKSOURCE_HSI: SPI5 clock = HSI
 - RCC_SPI5CLKSOURCE_CSI: SPI5 clock = CSI

- RCC_SPI5CLKSOURCE_HSE: SPI5 clock = HSE

Return value:

- None

[__HAL_RCC_GET_SPI5_SOURCE](#)**Description:**

- Macro to get the SPI5 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SPI5CLKSOURCE_D2PCLK1:SPI5 clock = D2PCLK1
 - RCC_SPI5CLKSOURCE_PLL2: SPI5 clock = PLL2
 - RCC_SPI5CLKSOURCE_PLL3: SPI5 clock = PLL3
 - RCC_SPI5CLKSOURCE_HSI: SPI5 clock = HSI
 - RCC_SPI5CLKSOURCE_CSI: SPI5 clock = CSI
 - RCC_SPI5CLKSOURCE_HSE: SPI5 clock = HSE

[__HAL_RCC_SPI6_CONFIG](#)**Description:**

- Macro to Configure the SPI6 clock source.

Parameters:

- __RCC_SPI6CLKSource__: defines the SPI6 clock source. This clock is derived from system PCLK, PLL2, PLL3, OSC This parameter can be one of the following values:
 - RCC_SPI6CLKSOURCE_D3PCLK1:SPI6 clock = D2PCLK1
 - RCC_SPI6CLKSOURCE_PLL2: SPI6 clock = PLL2
 - RCC_SPI6CLKSOURCE_PLL3: SPI6 clock = PLL3
 - RCC_SPI6CLKSOURCE_HSI: SPI6 clock = HSI
 - RCC_SPI6CLKSOURCE_CSI: SPI6 clock = CSI
 - RCC_SPI6CLKSOURCE_HSE: SPI6 clock = HSE

Return value:

- None

[__HAL_RCC_GET_SPI6_SOURCE](#)**Description:**

- Macro to get the SPI6 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_SPI6CLKSOURCE_D3PCLK1:SPI6 clock = D2PCLK1
 - RCC_SPI6CLKSOURCE_PLL2: SPI6 clock = PLL2
 - RCC_SPI6CLKSOURCE_PLL3: SPI6 clock = PLL3
 - RCC_SPI6CLKSOURCE_HSI: SPI6 clock = HSI
 - RCC_SPI6CLKSOURCE_CSI: SPI6 clock = CSI
 - RCC_SPI6CLKSOURCE_HSE: SPI6 clock = HSE

[__HAL_RCC_SDMMC_CONFIG](#)**Description:**

- Macro to configure the SDMMC clock.

Parameters:

- `__SDMMCCLKSource__`: specifies clock source for SDMMC. This parameter can be one of the following values:
 - `RCC_SDMMCCLKSOURCE_PLL`: PLLQ selected as SDMMC clock
 - `RCC_SDMMCCLKSOURCE_PLL2`: PLL2R selected as SDMMC clock

[_HAL_RCC_GET_SDMMC_SOURCE](#)

[_HAL_RCC_RNG_CONFIG](#)

Description:

- macro to configure the RNG clock (RNGCLK).

Parameters:

- `__RNGCLKSource__`: specifies the RNG clock source. This parameter can be one of the following values:
 - `RCC RNGCLKSOURCE_HSI48`: HSI48 selected as RNG clock
 - `RCC RNGCLKSOURCE_PLL`: PLL1Q selected as RNG clock
 - `RCC RNGCLKSOURCE_LSE`: LSE selected as RNG clock
 - `RCC RNGCLKSOURCE_LSI`: LSI selected as RNG clock

[_HAL_RCC_GET_RNG_SOURCE](#)

Description:

- macro to get the RNG clock source.

Return value:

- The: clock source can be one of the following values:
 - `RCC RNGCLKSOURCE_HSI48`: HSI48 selected as RNG clock
 - `RCC RNGCLKSOURCE_PLL`: PLL1Q selected as RNG clock
 - `RCC RNGCLKSOURCE_LSE`: LSE selected as RNG clock
 - `RCC RNGCLKSOURCE_LSI`: LSI selected as RNG clock

[_HAL_RCC_CRS_ENABLE_IT](#)

Description:

- Enable the specified CRS interrupts.

Parameters:

- `__INTERRUPT__`: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the following values:
 - `RCC CRS_IT_SYNCOK` SYNC event OK interrupt
 - `RCC CRS_IT_SYNCWARN` SYNC warning interrupt
 - `RCC CRS_IT_ERR` Synchronization or trimming error interrupt
 - `RCC CRS_IT_ESYNC` Expected SYNC interrupt

Return value:

- None

[_HAL_RCC_CRS_DISABLE_IT](#)

Description:

- Disable the specified CRS interrupts.

Parameters:

- `__INTERRUPT__`: specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
 - `RCC CRS_IT_SYNCOK` SYNC event OK interrupt
 - `RCC CRS_IT_SYNCWARN` SYNC warning interrupt

- RCC_CRS_IT_ERR Synchronization or trimming error interrupt
- RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- None

[__HAL_RCC_CRS_GET_IT_SOURCE](#)**Description:**

- Check whether the CRS interrupt has occurred or not.

Parameters:

- _INTERRUPT_: specifies the CRS interrupt source to check. This parameter can be one of the following values:
 - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
 - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
 - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
 - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:

- The: new state of _INTERRUPT_ (SET or RESET).

[RCC_CRS_IT_ERROR_MASK](#)**Description:**

- Clear the CRS interrupt pending bits.

Parameters:

- _INTERRUPT_: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
 - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
 - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
 - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
 - RCC_CRS_IT_ESYNC Expected SYNC interrupt
 - RCC_CRS_IT_TRIMOVF Trimming overflow or underflow interrupt
 - RCC_CRS_IT_SYNCERR SYNC error interrupt
 - RCC_CRS_IT_SYNCMISS SYNC missed interrupt

[__HAL_RCC_CRS_CLEAR_IT](#)[__HAL_RCC_CRS_GET_FLAG](#)**Description:**

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

Parameters:

- _FLAG_: specifies the flag to check. This parameter can be one of the following values:
 - RCC_CRS_FLAG_SYNCOK SYNC event OK
 - RCC_CRS_FLAG_SYNCWARN SYNC warning
 - RCC_CRS_FLAG_ERR Error
 - RCC_CRS_FLAG_ESYNC Expected SYNC
 - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
 - RCC_CRS_FLAG_SYNCERR SYNC error
 - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

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

RCC_CRS_FLAG_ERROR_MASK

Description:

- Clear the CRS specified FLAG.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - RCC_CRS_FLAG_SYNCOK SYNC event OK
 - RCC_CRS_FLAG_SYNCWARN SYNC warning
 - RCC_CRS_FLAG_ERR Error
 - RCC_CRS_FLAG_ESYNC Expected SYNC
 - RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
 - RCC_CRS_FLAG_SYNCERR SYNC error
 - RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- None

Notes:

- RCC_CRS_FLAG_ERR clears RCC_CRS_FLAG_TRIMOVF, RCC_CRS_FLAG_SYNCERR, RCC_CRS_FLAG_SYNCMISS and consequently RCC_CRS_FLAG_ERR

_HAL_RCC_CRS_CLEAR_FLAG

RCCEEx FDCAN Clock Source

RCC_FDCANCLKSOURCE_HSE

RCC_FDCANCLKSOURCE_PLL

RCC_FDCANCLKSOURCE_PLL2

RCCEEx FMC Clock Source

RCC_FMCCCLKSOURCE_D1HCLK

RCC_FMCCCLKSOURCE_PLL

RCC_FMCCCLKSOURCE_PLL2

RCC_FMCCCLKSOURCE_CLKP

RCC Extended HRTIM1 Clock Source

RCC_HRTIM1CLK_TIMCLK

RCC_HRTIM1CLK_CPUCLK

RCC Extended HRTIMx Clock Config

_HAL_RCC_HRTIM1_CONFIG

Description:

- Macro to configure the HRTIM1 prescaler clock source.

Parameters:

- __HRTIM1CLKSource__: specifies the HRTIM1 prescaler clock source. This parameter can be one of the following values:
 - RCC_HRTIM1CLK_TIMCLK Timers clock selected as HRTIM1 prescaler clock
 - RCC_HRTIM1CLK_CPUCLK CPU Clock selected as HRTIM1 clock

[__HAL_RCC_GET_HRTIM1_SOURCE](#)

Description:

- Macro to get the HRTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_HRTIM1CLK_TIMCLK Timers clock selected as HRTIM1 prescaler clock
 - RCC_HRTIM1CLK_CPUCLK CPU Clock selected as HRTIM1 clock

[__HAL_RCC_TIMCLKPRESCALER](#)

Description:

- Macro to configure the Timers clocks prescalers.

Parameters:

- __PRESC__: specifies the Timers clocks prescalers selection This parameter can be one of the following values:
 - RCC_TIMPRES_DESACTIVATED: The Timers kernels clocks prescaler is equal to rcc_hclk1 if D2PPREx is corresponding to division by 1 or 2, else it is equal to 2 x Frcc_pclkx_d2 (default after reset)
 - RCC_TIMPRES_ACTIVATED: The Timers kernels clocks prescaler is equal to rcc_hclk1 if D2PPREx is corresponding to division by 1, 2 or 4, else it is equal to 4 x Frcc_pclkx_d2

RCCEEx I2C1/2/3 Clock Source

[RCC_I2C123CLKSOURCE_D2PCLK1](#)

[RCC_I2C123CLKSOURCE_PLL3](#)

[RCC_I2C123CLKSOURCE_HSI](#)

[RCC_I2C123CLKSOURCE_CSI](#)

RCCEEx I2C1 Clock Source

[RCC_I2C1CLKSOURCE_D2PCLK1](#)

[RCC_I2C1CLKSOURCE_PLL3](#)

[RCC_I2C1CLKSOURCE_HSI](#)

[RCC_I2C1CLKSOURCE_CSI](#)

RCCEEx I2C2 Clock Source

[RCC_I2C2CLKSOURCE_D2PCLK1](#)

[RCC_I2C2CLKSOURCE_PLL3](#)

[RCC_I2C2CLKSOURCE_HSI](#)

[RCC_I2C2CLKSOURCE_CSI](#)

RCCEEx I2C3 Clock Source

[RCC_I2C3CLKSOURCE_D2PCLK1](#)

[RCC_I2C3CLKSOURCE_PLL3](#)

RCC_I2C3CLKSOURCE_HSI

RCC_I2C3CLKSOURCE_CSI

RCCEEx I2C4 Clock Source

RCC_I2C4CLKSOURCE_D3PCLK1

RCC_I2C4CLKSOURCE_PLL3

RCC_I2C4CLKSOURCE_HSI

RCC_I2C4CLKSOURCE_CSI

RCC Private macros to check input parameters

IS_RCC_PLL2CLOCKOUT_VALUE

IS_RCC_PLL3CLOCKOUT_VALUE

IS_RCC_USART16CLKSOURCE

IS_RCC_USART234578CLKSOURCE

IS_RCC_USART1CLKSOURCE

IS_RCC_USART2CLKSOURCE

IS_RCC_USART3CLKSOURCE

IS_RCC_UART4CLKSOURCE

IS_RCC_UART5CLKSOURCE

IS_RCC_USART6CLKSOURCE

IS_RCC_UART7CLKSOURCE

IS_RCC_UART8CLKSOURCE

IS_RCC_LPUART1CLKSOURCE

IS_RCC_I2C123CLKSOURCE

IS_RCC_I2C1CLKSOURCE

IS_RCC_I2C2CLKSOURCE

IS_RCC_I2C3CLKSOURCE

IS_RCC_I2C4CLKSOURCE

IS_RCC RNGCLKSOURCE

IS_RCC_HRTIM1CLKSOURCE

IS_RCC_USBCLKSOURCE

IS_RCC_USBCLKSOURCE

IS_RCC_SAI1CLK

IS_RCC_SAI23CLK

IS_RCC_SAI2CLK

IS_RCC_SAI3CLK

IS_RCC_SPI123CLK

IS_RCC_SPI1CLK

IS_RCC_SPI2CLK

IS_RCC_SPI3CLK

IS_RCC_SPI45CLK

IS_RCC_SPI4CLK

IS_RCC_SPI5CLK

IS_RCC_SPI6CLK

IS_RCC_SAI4ACLK

IS_RCC_SAI4BCLK

IS_RCC_PLL3M_VALUE

IS_RCC_PLL3N_VALUE

IS_RCC_PLL3P_VALUE

IS_RCC_PLL3Q_VALUE

IS_RCC_PLL3R_VALUE

IS_RCC_PLL2M_VALUE

IS_RCC_PLL2N_VALUE

IS_RCC_PLL2P_VALUE

IS_RCC_PLL2Q_VALUE

IS_RCC_PLL2R_VALUE

IS_RCC_LPTIM1CLK

IS_RCC_LPTIM2CLK

IS_RCC_LPTIM345CLK

IS_RCC_LPTIM3CLK
IS_RCC_LPTIM4CLK
IS_RCC_LPTIM5CLK
IS_RCC_QSPICLK
IS_RCC_FMCLK
IS_RCC_FDCANCLK
IS_RCC_SDMMC
IS_RCC_ADCCLKSOURCE
IS_RCC_SWPMI1CLKSOURCE
IS_RCC_DFSDM1CLKSOURCE
IS_RCC_SPDIFRXCLKSOURCE
IS_RCC_CECCLKSOURCE
IS_RCC_CLKPSOURCE
IS_RCC_TIMPRES
IS_RCC_SCOPE_WWDG
IS_RCC_CRS_SYNC_SOURCE
IS_RCC_CRS_SYNC_DIV
IS_RCC_CRS_SYNC_POLARITY
IS_RCC_CRS_RELOADVALUE
IS_RCC_CRS_ERRORLIMIT
IS_RCC_CRS_HSI48CALIBRATION
IS_RCC_CRS_FREQERRORDIR
RCCEx LPTIM1 Clock Source
RCC_LPTIM1CLKSOURCE_D2PCLK1
RCC_LPTIM1CLKSOURCE_PLL2
RCC_LPTIM1CLKSOURCE_PLL3
RCC_LPTIM1CLKSOURCE_LSE
RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_CLKP

RCCEEx LPTIM2 Clock Source

RCC_LPTIM2CLKSOURCE_D3PCLK1

RCC_LPTIM2CLKSOURCE_PLL2

RCC_LPTIM2CLKSOURCE_PLL3

RCC_LPTIM2CLKSOURCE_LSE

RCC_LPTIM2CLKSOURCE_LSI

RCC_LPTIM2CLKSOURCE_CLKP

RCCEEx LPTIM3/4/5 Clock Source

RCC_LPTIM345CLKSOURCE_D3PCLK1

RCC_LPTIM345CLKSOURCE_PLL2

RCC_LPTIM345CLKSOURCE_PLL3

RCC_LPTIM345CLKSOURCE_LSE

RCC_LPTIM345CLKSOURCE_LSI

RCC_LPTIM345CLKSOURCE_CLKP

RCCEEx LPTIM3 Clock Source

RCC_LPTIM3CLKSOURCE_D3PCLK1

RCC_LPTIM3CLKSOURCE_PLL2

RCC_LPTIM3CLKSOURCE_PLL3

RCC_LPTIM3CLKSOURCE_LSE

RCC_LPTIM3CLKSOURCE_LSI

RCC_LPTIM3CLKSOURCE_CLKP

RCCEEx LPTIM4 Clock Source

RCC_LPTIM4CLKSOURCE_D3PCLK1

RCC_LPTIM4CLKSOURCE_PLL2

RCC_LPTIM4CLKSOURCE_PLL3

RCC_LPTIM4CLKSOURCE_LSE

RCC_LPTIM4CLKSOURCE_LSI

RCC_LPTIM4CLKSOURCE_CLKP

RCCEEx LPTIM5 Clock Source

RCC_LPTIM5CLKSOURCE_D3PCLK1

RCC_LPTIM5CLKSOURCE_PLL2

RCC_LPTIM5CLKSOURCE_PLL3

RCC_LPTIM5CLKSOURCE_LSE

RCC_LPTIM5CLKSOURCE_LSI

RCC_LPTIM5CLKSOURCE_CLKP

RCCEEx LPUART1 Clock Source

RCC_LPUART1CLKSOURCE_D3PCLK1

RCC_LPUART1CLKSOURCE_PLL2

RCC_LPUART1CLKSOURCE_PLL3

RCC_LPUART1CLKSOURCE_HSI

RCC_LPUART1CLKSOURCE_CSI

RCC_LPUART1CLKSOURCE_LSE

RCCEEx Periph Clock Selection

RCC_PERIPHCLK_USART16

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART6

RCC_PERIPHCLK_USART234578

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_USART3

RCC_PERIPHCLK_UART4

RCC_PERIPHCLK_UART5

RCC_PERIPHCLK_UART7

RCC_PERIPHCLK_UART8

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_I2C123

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_I2C3

RCC_PERIPHCLK_I2C4
RCC_PERIPHCLK_LPTIM1
RCC_PERIPHCLK_LPTIM2
RCC_PERIPHCLK_LPTIM345
RCC_PERIPHCLK_LPTIM3
RCC_PERIPHCLK_LPTIM4
RCC_PERIPHCLK_LPTIM5
RCC_PERIPHCLK_SAI1
RCC_PERIPHCLK_SAI23
RCC_PERIPHCLK_SAI2
RCC_PERIPHCLK_SAI3
RCC_PERIPHCLK_SAI4A
RCC_PERIPHCLK_SAI4B
RCC_PERIPHCLK_SPI123
RCC_PERIPHCLK_SPI1
RCC_PERIPHCLK_SPI2
RCC_PERIPHCLK_SPI3
RCC_PERIPHCLK_SPI45
RCC_PERIPHCLK_SPI4
RCC_PERIPHCLK_SPI5
RCC_PERIPHCLK_SPI6
RCC_PERIPHCLK_FDCAN
RCC_PERIPHCLK_SDMMC
RCC_PERIPHCLK_RNG
RCC_PERIPHCLK_USB
RCC_PERIPHCLK_ADC
RCC_PERIPHCLK_SWPMI1
RCC_PERIPHCLK_DFSDM1

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_CEC

RCC_PERIPHCLK_FMC

RCC_PERIPHCLK_QSPI

RCC_PERIPHCLK_DSI

RCC_PERIPHCLK_SPDIFRX

RCC_PERIPHCLK_HRTIM1

RCC_PERIPHCLK_LTDC

RCC_PERIPHCLK_TIM

RCC_PERIPHCLK_CKPER

RCCEEx QSPI Clock Source

RCC_QSPICLKSOURCE_D1HCLK

RCC_QSPICLKSOURCE_PLL

RCC_QSPICLKSOURCE_PLL2

RCC_QSPICLKSOURCE_CLKP

RCCEEx RCC WWDGx

RCC_WWDG1

RCCEEx RNG Clock Source

RCC_RNGCLKSOURCE_HSI48

RCC_RNGCLKSOURCE_PLL

RCC_RNGCLKSOURCE_LSE

RCC_RNGCLKSOURCE_LSI

SAI1 Clock Source

RCC_SAI1CLKSOURCE_PLL

RCC_SAI1CLKSOURCE_PLL2

RCC_SAI1CLKSOURCE_PLL3

RCC_SAI1CLKSOURCE_PIN

RCC_SAI1CLKSOURCE_CLKP

SAI2/3 Clock Source

RCC_SAI23CLKSOURCE_PLL

RCC_SAI23CLKSOURCE_PLL2

RCC_SAI23CLKSOURCE_PLL3

RCC_SAI23CLKSOURCE_PIN

RCC_SAI23CLKSOURCE_CLKP

SAI2 Clock Source

RCC_SAI2CLKSOURCE_PLL

RCC_SAI2CLKSOURCE_PLL2

RCC_SAI2CLKSOURCE_PLL3

RCC_SAI2CLKSOURCE_PIN

RCC_SAI2CLKSOURCE_CLKP

SAI3 Clock Source

RCC_SAI3CLKSOURCE_PLL

RCC_SAI3CLKSOURCE_PLL2

RCC_SAI3CLKSOURCE_PLL3

RCC_SAI3CLKSOURCE_PIN

RCC_SAI3CLKSOURCE_CLKP

SAI4A Clock Source

RCC_SAI4ACLKSOURCE_PLL

RCC_SAI4ACLKSOURCE_PLL2

RCC_SAI4ACLKSOURCE_PLL3

RCC_SAI4ACLKSOURCE_PIN

RCC_SAI4ACLKSOURCE_CLKP

SAI4B Clock Source

RCC_SAI4BCLKSOURCE_PLL

RCC_SAI4BCLKSOURCE_PLL2

RCC_SAI4BCLKSOURCE_PLL3

RCC_SAI4BCLKSOURCE_PIN

RCC_SAI4BCLKSOURCE_CLKP

RCCEEx SDMMC Clock Source

RCC_SDMMCCLKSOURCE_PLL

RCC_SDMMCCLKSOURCE_PLL2

RCCEEx SPDIFRX Clock Source

RCC_SPDIFRXCLKSOURCE_PLL

RCC_SPDIFRXCLKSOURCE_PLL2

RCC_SPDIFRXCLKSOURCE_PLL3

RCC_SPDIFRXCLKSOURCE_HSI

SPI1/2/3 Clock Source

RCC_SPI123CLKSOURCE_PLL

RCC_SPI123CLKSOURCE_PLL2

RCC_SPI123CLKSOURCE_PLL3

RCC_SPI123CLKSOURCE_PIN

RCC_SPI123CLKSOURCE_CLKP

SPI1 Clock Source

RCC_SPI1CLKSOURCE_PLL

RCC_SPI1CLKSOURCE_PLL2

RCC_SPI1CLKSOURCE_PLL3

RCC_SPI1CLKSOURCE_PIN

RCC_SPI1CLKSOURCE_CLKP

SPI2 Clock Source

RCC_SPI2CLKSOURCE_PLL

RCC_SPI2CLKSOURCE_PLL2

RCC_SPI2CLKSOURCE_PLL3

RCC_SPI2CLKSOURCE_PIN

RCC_SPI2CLKSOURCE_CLKP

SPI3 Clock Source

RCC_SPI3CLKSOURCE_PLL

RCC_SPI3CLKSOURCE_PLL2

RCC_SPI3CLKSOURCE_PLL3

RCC_SPI3CLKSOURCE_PIN

RCC_SPI3CLKSOURCE_CLKP

SPI4/5 Clock Source`RCC_SPI45CLKSOURCE_D2PCLK1``RCC_SPI45CLKSOURCE_PLL2``RCC_SPI45CLKSOURCE_PLL3``RCC_SPI45CLKSOURCE_HSI``RCC_SPI45CLKSOURCE_CSI``RCC_SPI45CLKSOURCE_HSE`***SPI4 Clock Source***`RCC_SPI4CLKSOURCE_D2PCLK1``RCC_SPI4CLKSOURCE_PLL2``RCC_SPI4CLKSOURCE_PLL3``RCC_SPI4CLKSOURCE_HSI``RCC_SPI4CLKSOURCE_CSI``RCC_SPI4CLKSOURCE_HSE`***SPI5 Clock Source***`RCC_SPI5CLKSOURCE_D2PCLK1``RCC_SPI5CLKSOURCE_PLL2``RCC_SPI5CLKSOURCE_PLL3``RCC_SPI5CLKSOURCE_HSI``RCC_SPI5CLKSOURCE_CSI``RCC_SPI5CLKSOURCE_HSE`***SPI6 Clock Source***`RCC_SPI6CLKSOURCE_D3PCLK1``RCC_SPI6CLKSOURCE_PLL2``RCC_SPI6CLKSOURCE_PLL3``RCC_SPI6CLKSOURCE_HSI``RCC_SPI6CLKSOURCE_CSI``RCC_SPI6CLKSOURCE_HSE`***RCCEEx SWPMI1 Clock Source***`RCC_SWPMI1CLKSOURCE_D2PCLK1`

RCC_SWPMI1CLKSOURCE_HSI

RCCEEx TIM Prescaler Selection

RCC_TIMPRES_DESACTIVATED

RCC_TIMPRES_ACTIVATED

RCCEEx UART4 Clock Source

RCC_UART4CLKSOURCE_D2PCLK1

RCC_UART4CLKSOURCE_PLL2

RCC_UART4CLKSOURCE_PLL3

RCC_UART4CLKSOURCE_HSI

RCC_UART4CLKSOURCE_CSI

RCC_UART4CLKSOURCE_LSE

RCCEEx UART5 Clock Source

RCC_UART5CLKSOURCE_D2PCLK1

RCC_UART5CLKSOURCE_PLL2

RCC_UART5CLKSOURCE_PLL3

RCC_UART5CLKSOURCE_HSI

RCC_UART5CLKSOURCE_CSI

RCC_UART5CLKSOURCE_LSE

RCCEEx UART7 Clock Source

RCC_UART7CLKSOURCE_D2PCLK1

RCC_UART7CLKSOURCE_PLL2

RCC_UART7CLKSOURCE_PLL3

RCC_UART7CLKSOURCE_HSI

RCC_UART7CLKSOURCE_CSI

RCC_UART7CLKSOURCE_LSE

RCCEEx UART8 Clock Source

RCC_UART8CLKSOURCE_D2PCLK1

RCC_UART8CLKSOURCE_PLL2

RCC_UART8CLKSOURCE_PLL3

RCC_UART8CLKSOURCE_HSI

RCC_UART8CLKSOURCE_CSI

RCC_UART8CLKSOURCE_LSE

RCCEEx USART1/6 Clock Source

RCC_USART16CLKSOURCE_D2PCLK2

RCC_USART16CLKSOURCE_PLL2

RCC_USART16CLKSOURCE_PLL3

RCC_USART16CLKSOURCE_HSI

RCC_USART16CLKSOURCE_CSI

RCC_USART16CLKSOURCE_LSE

RCCEEx USART1 Clock Source

RCC_USART1CLKSOURCE_D2PCLK2

RCC_USART1CLKSOURCE_PLL2

RCC_USART1CLKSOURCE_PLL3

RCC_USART1CLKSOURCE_HSI

RCC_USART1CLKSOURCE_CSI

RCC_USART1CLKSOURCE_LSE

RCCEEx USART2/3/4/5/7/8 Clock Source

RCC_USART234578CLKSOURCE_D2PCLK1

RCC_USART234578CLKSOURCE_PLL2

RCC_USART234578CLKSOURCE_PLL3

RCC_USART234578CLKSOURCE_HSI

RCC_USART234578CLKSOURCE_CSI

RCC_USART234578CLKSOURCE_LSE

RCCEEx USART2 Clock Source

RCC_USART2CLKSOURCE_D2PCLK1

RCC_USART2CLKSOURCE_PLL2

RCC_USART2CLKSOURCE_PLL3

RCC_USART2CLKSOURCE_HSI

RCC_USART2CLKSOURCE_CSI

RCC_USART2CLKSOURCE_LSE

RCCEEx USART3 Clock Source

RCC_USART3CLKSOURCE_D2PCLK1

RCC_USART3CLKSOURCE_PLL2

RCC_USART3CLKSOURCE_PLL3

RCC_USART3CLKSOURCE_HSI

RCC_USART3CLKSOURCE_CSI

RCC_USART3CLKSOURCE_LSE

RCCEEx USART6 Clock Source

RCC_USART6CLKSOURCE_D2PCLK2

RCC_USART6CLKSOURCE_PLL2

RCC_USART6CLKSOURCE_PLL3

RCC_USART6CLKSOURCE_HSI

RCC_USART6CLKSOURCE_CSI

RCC_USART6CLKSOURCE_LSE

57 HAL RNG Generic Driver

57.1 RNG Firmware driver registers structures

57.1.1 RNG_InitTypeDef

Data Fields

- *uint32_t ClockErrorDetection*

Field Documentation

- *uint32_t RNG_InitTypeDef::ClockErrorDetection*

CED Clock error detection

57.1.2 RNG_HandleTypeDef

Data Fields

- *RNG_TypeDef * Instance*
- *RNG_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_RNG_StateTypeDef State*
- *uint32_t RandomNumber*

Field Documentation

- *RNG_TypeDef* RNG_HandleTypeDef::Instance*

Register base address

- *RNG_InitTypeDef RNG_HandleTypeDef::Init*

RNG parameters

- *HAL_LockTypeDef RNG_HandleTypeDef::Lock*

RNG locking object

- *__IO HAL_RNG_StateTypeDef RNG_HandleTypeDef::State*

RNG communication state

- *uint32_t RNG_HandleTypeDef::RandomNumber*

Last Generated RNG Data

57.2 RNG Firmware driver API description

57.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using `__HAL_RCC_RNG_CLK_ENABLE()` macro in `HAL_RNG_MspInit()`.
2. Activate the RNG peripheral using `HAL_RNG_Init()` function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using `HAL_RNG_GenerateRandomNumber()` function.

57.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the RNG_InitTypeDef and create the associated handle
- Deinitialize the RNG peripheral
- Initialize the RNG MSP
- Deinitialize RNG MSP

This section contains the following APIs:

- [*HAL_RNG_Init*](#)
- [*HAL_RNG_DelInit*](#)
- [*HAL_RNG_MspInit*](#)
- [*HAL_RNG_MspDelInit*](#)

57.2.3 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- [*HAL_RNG_GenerateRandomNumber*](#)
- [*HAL_RNG_GenerateRandomNumber_IT*](#)
- [*HAL_RNG_IRQHandler*](#)
- [*HAL_RNG_ReadLastRandomNumber*](#)
- [*HAL_RNG_ReadyDataCallback*](#)
- [*HAL_RNG_ErrorCallback*](#)

57.2.4 Peripheral State functions

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

This section contains the following APIs:

- [*HAL_RNG_GetState*](#)

57.2.5 Detailed description of functions

[*HAL_RNG_Init*](#)

Function name

`HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)`

Function description

Initializes the RNG peripheral and creates the associated handle.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_DelInit

Function name

`HAL_StatusTypeDef HAL_RNG_DelInit (RNG_HandleTypeDef * hrng)`

Function description

DeInitializes the RNG peripheral.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_MspInit

Function name

`void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)`

Function description

Initializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_MspDelInit

Function name

`void HAL_RNG_MspDelInit (RNG_HandleTypeDef * hrng)`

Function description

DeInitializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_GenerateRandomNumber

Function name

`HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)`

Function description

Generates a 32-bit random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** pointer to generated random number variable if successful.

Return values

- **HAL:** status

Notes

- Each time the random number data is read the RNG_FLAG_DRDY flag is automatically cleared.

HAL_RNG_GenerateRandomNumber_IT

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)

Function description

Generates a 32-bit random number in interrupt mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_ReadLastRandomNumber

Function name

uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)

Function description

Read latest generated random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **random:** value

HAL_RNG_IRQHandler

Function name

void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)

Function description

Handles RNG interrupt request.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

Notes

- In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using __HAL_RNG_CLEAR_IT(). The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used.
- In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using __HAL_RNG_CLEAR_IT(), then disable and enable the RNG peripheral to reinitialize and restart the RNG.

- User-written HAL_RNG_ErrorCallback() API is called once whether SEIS or CEIS are set.

HAL_RNG_ErrorCallback

Function name

void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)

Function description

RNG error callbacks.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_ReadyDataCallback

Function name

void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)

Function description

Data Ready callback in non-blocking mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** generated random number.

Return values

- **None:**

HAL_RNG_GetState

Function name

HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)

Function description

Returns the RNG state.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** state

57.3 RNG Firmware driver defines

57.3.1 RNG

RNG Interrupt definition

RNG_IT_DRDY

Data Ready interrupt

RNG_IT_CEI

Clock error interrupt

RNG_IT_SEI

Seed error interrupt

RNG Flag definition**RNG_FLAG_DRDY**

Data ready

RNG_FLAG_CECS

Clock error current status

RNG_FLAG_SECS

Seed error current status

RNG Clock Error Detection**RNG_CED_ENABLE**

Clock error detection Enabled

RNG_CED_DISABLE

Clock error detection Disabled

RNG Exported Macros**_HAL_RNG_RESET_HANDLE_STATE****Description:**

- Reset RNG handle state.

Parameters:

- _HANDLE_: RNG Handle

Return value:

- None

_HAL_RNG_ENABLE**Description:**

- Enables the RNG peripheral.

Parameters:

- _HANDLE_: RNG Handle

Return value:

- None

_HAL_RNG_DISABLE**Description:**

- Disables the RNG peripheral.

Parameters:

- _HANDLE_: RNG Handle

Return value:

- None

_HAL_RNG_GET_FLAG**Description:**

- Check the selected RNG flag status.

Parameters:

- `__HANDLE__`: RNG Handle
- `__FLAG__`: RNG flag This parameter can be one of the following values:
 - `RNG_FLAG_DRDY`: Data ready
 - `RNG_FLAG_CECS`: Clock error current status
 - `RNG_FLAG_SECS`: Seed error current status

Return value:

- The new state of `__FLAG__` (SET or RESET).

[`__HAL_RNG_CLEAR_FLAG`](#)**Description:**

- Clears the selected RNG flag status.

Parameters:

- `__HANDLE__`: RNG handle
- `__FLAG__`: RNG flag to clear

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment, flags `RNG_FLAG_DRDY`, `RNG_FLAG_CECS` and `RNG_FLAG_SECS` are read-only.

[`__HAL_RNG_ENABLE_IT`](#)**Description:**

- Enables the RNG interrupts.

Parameters:

- `__HANDLE__`: RNG Handle

Return value:

- None

[`__HAL_RNG_DISABLE_IT`](#)**Description:**

- Disables the RNG interrupts.

Parameters:

- `__HANDLE__`: RNG Handle

Return value:

- None

[`__HAL_RNG_GET_IT`](#)**Description:**

- Checks whether the specified RNG interrupt has occurred or not.

Parameters:

- `__HANDLE__`: RNG Handle
- `__INTERRUPT__`: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
 - `RNG_IT_DRDY`: Data ready interrupt
 - `RNG_IT_CEI`: Clock error interrupt

- RNG_IT_SEI: Seed error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

[__HAL_RNG_CLEAR_IT](#)**Description:**

- Clear the RNG interrupt status flags.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
 - RNG_IT_CEI: Clock error interrupt
 - RNG_IT_SEI: Seed error interrupt

Return value:

- None

Notes:

- RNG_IT_DRDY flag is read-only, reading RNG_DR register automatically clears RNG_IT_DRDY.

58 HAL RTC Generic Driver

58.1 RTC Firmware driver registers structures

58.1.1 RTC_InitTypeDef

Data Fields

- `uint32_t HourFormat`
- `uint32_t AsynchPrediv`
- `uint32_t SynchPrediv`
- `uint32_t OutPut`
- `uint32_t OutPutRemap`
- `uint32_t OutPutPolarity`
- `uint32_t OutPutType`

Field Documentation

- `uint32_t RTC_InitTypeDef::HourFormat`

Specifies the RTC Hour Format. This parameter can be a value of [RTC Hour Formats](#)

- `uint32_t RTC_InitTypeDef::AsynchPrediv`

Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F

- `uint32_t RTC_InitTypeDef::SynchPrediv`

Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF

- `uint32_t RTC_InitTypeDef::OutPut`

Specifies which signal will be routed to the RTC output. This parameter can be a value of [RTC Output Selection Definitions](#)

- `uint32_t RTC_InitTypeDef::OutPutRemap`

Specifies the remap for RTC output. This parameter can be a value of [RTC Output ALARM OUT Remap](#)

- `uint32_t RTC_InitTypeDef::OutPutPolarity`

Specifies the polarity of the output signal. This parameter can be a value of [RTC Output Polarity Definitions](#)

- `uint32_t RTC_InitTypeDef::OutPutType`

Specifies the RTC Output Pin mode. This parameter can be a value of [RTC Output Type ALARM OUT](#)

58.1.2 RTC_TimeTypeDef

Data Fields

- `uint8_t Hours`
- `uint8_t Minutes`
- `uint8_t Seconds`
- `uint8_t TimeFormat`
- `uint32_t SubSeconds`
- `uint32_t SecondFraction`
- `uint32_t DayLightSaving`

- `uint32_t StoreOperation`

Field Documentation

- `uint8_t RTC_TimeTypeDef::Hours`

Specifies the RTC Time Hour. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the RTC_HourFormat_12 is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the RTC_HourFormat_24 is selected

- `uint8_t RTC_TimeTypeDef::Minutes`

Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59

- `uint8_t RTC_TimeTypeDef::Seconds`

Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59

- `uint8_t RTC_TimeTypeDef::TimeFormat`

Specifies the RTC AM/PM Time. This parameter can be a value of **RTC AM PM Definitions**

- `uint32_t RTC_TimeTypeDef::SubSeconds`

Specifies the RTC_SSR RTC Sub Second register content. This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity

- `uint32_t RTC_TimeTypeDef::SecondFraction`

Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function

- `uint32_t RTC_TimeTypeDef::DayLightSaving`

Specifies RTC_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of **RTC DayLight Saving Definitions**

- `uint32_t RTC_TimeTypeDef::StoreOperation`

Specifies RTC_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of **RTC Store Operation Definitions**

58.1.3

RTC_DateTypeDef

Data Fields

- `uint8_t WeekDay`
- `uint8_t Month`
- `uint8_t Date`
- `uint8_t Year`

Field Documentation

- `uint8_t RTC_DateTypeDef::WeekDay`

Specifies the RTC Date WeekDay. This parameter can be a value of **RTC WeekDay Definitions**

- `uint8_t RTC_DateTypeDef::Month`

Specifies the RTC Date Month (in BCD format). This parameter can be a value of **RTC Month Date Definitions**

- `uint8_t RTC_DateTypeDef::Date`

Specifies the RTC Date. This parameter must be a number between Min_Data = 1 and Max_Data = 31

- `uint8_t RTC_DateTypeDef::Year`

Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99

58.1.4 RTC_AlarmTypeDef

Data Fields

- *RTC_TimeTypeDef AlarmTime*
- *uint32_t AlarmMask*
- *uint32_t AlarmSubSecondMask*
- *uint32_t AlarmDateWeekDaySel*
- *uint8_t AlarmDateWeekDay*
- *uint32_t Alarm*

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
Specifies the RTC Alarm Masks. This parameter can be a value of **RTC Alarm Mask Definitions**
- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of **RTC Alarm Sub Seconds Masks Definitions**
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of **RTC Alarm Date WeekDay Definitions**
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of **RTC WeekDay Definitions**
- ***uint32_t RTC_AlarmTypeDef::Alarm***
Specifies the alarm . This parameter can be a value of **RTC Alarms Definitions**

58.1.5 RTC_HandleTypeDef

Data Fields

- *RTC_TypeDef * Instance*
- *RTC_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_RTCStateTypeDef State*

Field Documentation

- ***RTC_TypeDef* RTC_HandleTypeDef::Instance***
Register base address
- ***RTC_InitTypeDef RTC_HandleTypeDef::Init***
RTC required parameters
- ***HAL_LockTypeDef RTC_HandleTypeDef::Lock***
RTC locking object
- ***__IO HAL_RTCStateTypeDef RTC_HandleTypeDef::State***
Time communication state

58.2 RTC Firmware driver API description

58.2.1 RTC Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

58.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

58.2.3 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

1. Call the function HAL_RCCEx_PeriphCLKConfig with RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or any HSE divider)
2. Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() macro.

58.2.4 How to use RTC Driver

1. Enable the RTC domain access (see description in the section above).
2. Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

Time and Date configuration

1. To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
2. To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

1. To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
2. To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

58.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.

58.2.6 Initialization and de-initialization functions

This section provide functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
 - A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
 - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers. The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- [HAL_RTC_Init](#)
- [HAL_RTC_DelInit](#)
- [HAL_RTC_MspInit](#)
- [HAL_RTC_MspDelInit](#)

58.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- [HAL_RTC_SetTime](#)
- [HAL_RTC_GetTime](#)
- [HAL_RTC_SetDate](#)
- [HAL_RTC_GetDate](#)

58.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- [HAL_RTC_SetAlarm](#)
- [HAL_RTC_SetAlarm_IT](#)
- [HAL_RTC_DeactivateAlarm](#)
- [HAL_RTC_GetAlarm](#)
- [HAL_RTC_AlarmIRQHandler](#)
- [HAL_RTC_AlarmAEventCallback](#)
- [HAL_RTC_PollForAlarmAEvent](#)

58.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- **HAL_RTC_WaitForSynchro**

58.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- **HAL_RTC_GetState**

58.2.11 Detailed description of functions

HAL_RTC_Init

Function name

HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC according to the specified parameters in the RTC_InitTypeDef structure and initialize the associated handle.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTC_DeInit

Function name

HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This function doesn't reset the RTC Backup Data registers.

HAL_RTC_MspInit

Function name

void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_MspDelInit

Function name

```
void HAL_RTC_MspDelInit (RTC_HandleTypeDef * hrtc)
```

Function description

DeInitialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_SetTime

Function name

```
HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
```

Function description

Set RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
 - RTC_FORMAT_BIN: Binary data format
 - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_GetTime

Function name

```
HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
```

Function description

Get RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation.
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
 - RTC_FORMAT_BIN: Binary data format
 - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS
- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.

HAL_RTC_SetDate

Function name

```
HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
```

Function description

Set RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
 - RTC_FORMAT_BIN: Binary data format
 - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_GetDate

Function name

```
HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
```

Function description

Get RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to Date structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
 - RTC_FORMAT_BIN: Binary data format
 - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

HAL_RTC_SetAlarm

Function name

```
HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,  
uint32_t Format)
```

Function description

Set the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
 - RTC_FORMAT_BIN: Binary data format
 - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_SetAlarm_IT

Function name

```
HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,  
uint32_t Format)
```

Function description

Set the specified RTC Alarm with Interrupt.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
 - RTC_FORMAT_BIN: Binary data format
 - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).
- The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

HAL_RTC_DeactivateAlarm

Function name

```
HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)
```

Function description

Deactivate the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle

- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
 - RTC_ALARM_A: AlarmA
 - RTC_ALARM_B: AlarmB

Return values

- **HAL:** status

HAL_RTC_GetAlarm

Function name

```
HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,  
uint32_t Alarm, uint32_t Format)
```

Function description

Get the RTC Alarm value and masks.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Date structure
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
 - RTC_ALARM_A: AlarmA
 - RTC_ALARM_B: AlarmB
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
 - RTC_FORMAT_BIN: Binary data format
 - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTC_AlarmIRQHandler

Function name

```
void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)
```

Function description

Handle Alarm interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_PollForAlarmAEvent

Function name

```
HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
```

Function description

Handle AlarmA Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTC_AlarmAEventCallback**Function name**

void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm A callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_WaitForSynchro**Function name**

HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)

Function description

Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Resynchronization mode is write protected, use the `__HAL_RTC_WRITEPROTECTION_DISABLE()` before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

HAL_RTC_GetState**Function name**

HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)

Function description

Return the RTC handle state.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** state

RTC_EnterInitMode**Function name**

HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)

Function description

Enter the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Initialization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.

RTC_ByteToBcd2

Function name

`uint8_t RTC_ByteToBcd2 (uint8_t Value)`

Function description

Convert a 2 digit decimal to BCD format.

Parameters

- **Value:** Byte to be converted

Return values

- **Converted:** byte

RTC_Bcd2ToByte

Function name

`uint8_t RTC_Bcd2ToByte (uint8_t Value)`

Function description

Convert from 2 digit BCD to Binary.

Parameters

- **Value:** BCD value to be converted

Return values

- **Converted:** word

58.3 RTC Firmware driver defines

58.3.1 RTC

RTC Alarm Date WeekDay Definitions

`RTC_ALARM_DATE_WEEKDAYSEL_DATE`

`RTC_ALARM_DATE_WEEKDAYSEL_WEEKDAY`

RTC Alarm Mask Definitions

`RTC_ALARM_MASK_NONE`

`RTC_ALARM_MASK_DATE_WEEKDAY`

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL

All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDMASK_SS14_1

SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.

RTC_ALARMSUBSECONDMASK_SS14_2

SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_3

SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_4

SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_5

SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_6

SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_7

SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_8

SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_9

SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_10

SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_11

SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_12

SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_13

SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDMASK_SS14

SS[14] is don't care in Alarm comparison.Only SS[13:0] are compared

RTC_ALARMSUBSECONDMASK_NONE

SS[14:0] are compared and must match to activate alarm.

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC DayLight Saving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros

_HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

Parameters:

- _HANDLE: RTC handle.

Return value:

- None

_HAL_RTC_WRITEPROTECTION_DISABLE

Description:

- Disable the write protection for RTC registers.

Parameters:

- _HANDLE: specifies the RTC handle.

Return value:

- None

_HAL_RTC_WRITEPROTECTION_ENABLE

Description:

- Enable the write protection for RTC registers.

Parameters:

- _HANDLE: specifies the RTC handle.

Return value:

- None

_HAL_RTC_ALARMA_ENABLE

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- _HANDLE: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_ALARMA_DISABLE](#)**Description:**

- Disable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_ALARM_B_ENABLE](#)**Description:**

- Enable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_ALARM_B_DISABLE](#)**Description:**

- Disable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_ALARM_ENABLE_IT](#)**Description:**

- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
 - RTC_IT_ALRA: Alarm A interrupt
 - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

[__HAL_RTC_ALARM_DISABLE_IT](#)**Description:**

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:

- RTC_IT_ALRA: Alarm A interrupt
- RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

[__HAL_RTC_ALARM_GET_IT](#)**Description:**

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
 - RTC_IT_ALRA: Alarm A interrupt
 - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

[__HAL_RTC_ALARM_GET_FLAG](#)**Description:**

- Get the selected RTC Alarm's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRBF
 - RTC_FLAG_ALRAWF
 - RTC_FLAG_ALRBWF

Return value:

- None

[__HAL_RTC_ALARM_CLEAR_FLAG](#)**Description:**

- Clear the RTC Alarm's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRBF

Return value:

- None

[__HAL_RTC_ALARM_GET_IT_SOURCE](#)**Description:**

- Check whether the specified RTC Alarm interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:

- RTC_IT_ALRA: Alarm A interrupt
- RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

[__HAL_RTC_ALARM_EXTI_ENABLE_IT](#)**Description:**

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

[__HAL_RTC_ALARM_EXTI_DISABLE_IT](#)**Description:**

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

[__HAL_RTC_ALARM_EXTI_ENABLE_EVENT](#)**Description:**

- Enable event on the RTC Alarm associated Exti line.

Return value:

- None.

[__HAL_RTC_ALARM_EXTI_DISABLE_EVENT](#)**Description:**

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None.

[__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE](#)**Description:**

- Enable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

[__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE](#)**Description:**

- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

[__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE](#)**Description:**

- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

[__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE](#)**Description:**

- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

[__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE](#)**Description:**

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

[__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE](#)**Description:**

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None

[__HAL_RTC_ALARM_EXTI_GET_FLAG](#)**Description:**

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

[__HAL_RTC_ALARM_EXTI_CLEAR_FLAG](#)**Description:**

- Clear the RTC Alarm associated Exti line flag.

Return value:

- None.

[__HAL_RTC_ALARM_EXTI_GENERATE_SWIT](#)**Description:**

- Generate a Software interrupt on RTC Alarm associated Exti line.

Return value:

- None

RTC Flags Definitions

[RTC_FLAG_RECALPF](#)

[RTC_FLAG_TAMP3F](#)

[RTC_FLAG_TAMP2F](#)

[RTC_FLAG_TAMP1F](#)

[RTC_FLAG_TSOVF](#)

[RTC_FLAG_TSF](#)

`RTC_FLAG_ITSF`

`RTC_FLAG_WUTF`

`RTC_FLAG_ALRBF`

`RTC_FLAG_ALRAF`

`RTC_FLAG_INITF`

`RTC_FLAG_RSF`

`RTC_FLAG_INITS`

`RTC_FLAG_SHPF`

`RTC_FLAG_WUTWF`

`RTC_FLAG_ALRBWF`

`RTC_FLAG_ALRAWF`

RTC Hour Formats

`RTC_HOURFORMAT_24`

`RTC_HOURFORMAT_12`

RTC Input Parameter Format Definitions

`RTC_FORMAT_BIN`

`RTC_FORMAT_BCD`

RTC Interrupts Definitions

`RTC_IT_TS`

Enable Timestamp Interrupt

`RTC_IT_WUT`

Enable Wakeup timer Interrupt

`RTC_IT_ALRA`

Enable Alarm A Interrupt

`RTC_IT_ALRB`

Enable Alarm B Interrupt

`RTC_IT_TAMP`

Enable all Tamper Interrupt

`RTC_IT_TAMP1`

Enable Tamper 1 Interrupt

`RTC_IT_TAMP2`

Enable Tamper 2 Interrupt

RTC_IT_TAMP3

Enable Tamper 3 Interrupt

RTC Private macros to check input parameters

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_REMAP

IS_RTC_HOURFORMAT12

IS_RTC_DAYLIGHT_SAVING

IS_RTC_STORE_OPERATION

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SECOND_VALUE

IS_RTC_ALARM_SUB_SECOND_MASK

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_OPENDRAIN

RTC_OUTPUT_TYPE_PUSHPULL

RTC Store Operation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

59 HAL RTC Extension Driver

59.1 RTCE Firmware driver registers structures

59.1.1 RTC_TamperTypeDef

Data Fields

- *uint32_t Tamper*
- *uint32_t Interrupt*
- *uint32_t Trigger*
- *uint32_t NoErase*
- *uint32_t MaskFlag*
- *uint32_t Filter*
- *uint32_t SamplingFrequency*
- *uint32_t PrechargeDuration*
- *uint32_t TamperPullUp*
- *uint32_t TimeStampOnTamperDetection*

Field Documentation

- *uint32_t RTC_TamperTypeDef::Tamper*

Specifies the Tamper Pin. This parameter can be a value of [RTC Tamper Pins Definitions](#)

- *uint32_t RTC_TamperTypeDef::Interrupt*

Specifies the Tamper Interrupt. This parameter can be a value of [RTC Tamper Interrupts Definitions](#)

- *uint32_t RTC_TamperTypeDef::Trigger*

Specifies the Tamper Trigger. This parameter can be a value of [RTC Tamper Triggers Definitions](#)

- *uint32_t RTC_TamperTypeDef::NoErase*

Specifies the Tamper no erase mode. This parameter can be a value of [RTC Tamper EraseBackUp Definitions](#)

- *uint32_t RTC_TamperTypeDef::MaskFlag*

Specifies the Tamper Flag masking. This parameter can be a value of [RTC Tamper Mask Flag Definitions](#)

- *uint32_t RTC_TamperTypeDef::Filter*

Specifies the RTC Filter Tamper. This parameter can be a value of [RTC Tamper Filter Definitions](#)

- *uint32_t RTC_TamperTypeDef::SamplingFrequency*

Specifies the sampling frequency. This parameter can be a value of [RTC Tamper Sampling Frequencies Definitions](#)

- *uint32_t RTC_TamperTypeDef::PrechargeDuration*

Specifies the Precharge Duration . This parameter can be a value of [RTC Tamper Pin Precharge Duration Definitions](#)

- *uint32_t RTC_TamperTypeDef::TamperPullUp*

Specifies the Tamper PullUp . This parameter can be a value of [RTC Tamper Pull Up Definitions](#)

- *uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection*

Specifies the TimeStampOnTamperDetection. This parameter can be a value of [RTC Tamper TimeStamp On Tamper Detection Definitions](#)

59.2 RTCEx Firmware driver API description

59.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEX_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEX_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEX_GetWakeUpTimer() function.

Outputs configuration

The RTC has 2 different outputs:

- RTC_ALARM: this output is used to manage the RTC Alarm A, Alarm B and WaKeUp signals. To output the selected RTC signal, use the HAL_RTC_Init() function.
- RTC_CALIB: this output is 512Hz signal or 1Hz. To enable the RTC_CALIB, use the HAL_RTCEX_SetCalibrationOutPut() function.
- Two pins can be used as RTC_ALARM or RTC_CALIB (PC13, PB2) managed on the RTC_OR register.
- When the RTC_CALIB or RTC_ALARM output is selected, the RTC_OUT pin is automatically configured in output alternate function.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s, 16s and 8s) using the HAL_RTCEX_SetSmoothCalib() function.

TimeStamp configuration

- Enable the RTC TimeStamp using the HAL_RTCEX_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEX_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEX_GetTimeStamp() function.

Internal TimeStamp configuration

- Enable the RTC internal TimeStamp using the HAL_RTCEX_SetInternalTimeStamp() function. User has to check internal timestamp occurrence using __HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEX_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL_RTCEX_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEX_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the RTC_TAMPCCR register.

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEX_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTCEX_BKUPRead() function.

59.2.2 RTC TimeStamp and Tamper functions

This section provide functions allowing to configure TimeStamp feature

This section contains the following APIs:

- [`HAL_RTCEx_SetTimeStamp`](#)
- [`HAL_RTCEx_SetTimeStamp_IT`](#)
- [`HAL_RTCEx_DeactivateTimeStamp`](#)
- [`HAL_RTCEx_SetInternalTimeStamp`](#)
- [`HAL_RTCEx_DeactivateInternalTimeStamp`](#)
- [`HAL_RTCEx_GetTimeStamp`](#)
- [`HAL_RTCEx_SetTamper`](#)
- [`HAL_RTCEx_SetTamper_IT`](#)
- [`HAL_RTCEx_DeactivateTamper`](#)
- [`HAL_RTCEx_TamperTimeStampIRQHandler`](#)
- [`HAL_RTCEx_TimeStampEventCallback`](#)
- [`HAL_RTCEx_Tamper1EventCallback`](#)
- [`HAL_RTCEx_Tamper2EventCallback`](#)
- [`HAL_RTCEx_Tamper3EventCallback`](#)
- [`HAL_RTCEx_PollForTimeStampEvent`](#)
- [`HAL_RTCEx_PollForTamper1Event`](#)
- [`HAL_RTCEx_PollForTamper2Event`](#)
- [`HAL_RTCEx_PollForTamper3Event`](#)

59.2.3 RTC Wake-up functions

This section provide functions allowing to configure Wake-up feature

This section contains the following APIs:

- [`HAL_RTCEx_SetWakeUpTimer`](#)
- [`HAL_RTCEx_SetWakeUpTimer_IT`](#)
- [`HAL_RTCEx_DeactivateWakeUpTimer`](#)
- [`HAL_RTCEx_GetWakeUpTimer`](#)
- [`HAL_RTCEx_WakeUpTimerIRQHandler`](#)
- [`HAL_RTCEx_WakeUpTimerEventCallback`](#)
- [`HAL_RTCEx_PollForWakeUpTimerEvent`](#)

59.2.4 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- [`HAL_RTCEx_BKUPWrite`](#)

- [**HAL_RTCEx_BKUPRead**](#)
- [**HAL_RTCEx_SetSmoothCalib**](#)
- [**HAL_RTCEx_SetSynchroShift**](#)
- [**HAL_RTCEx_SetCalibrationOutPut**](#)
- [**HAL_RTCEx_DeactivateCalibrationOutPut**](#)
- [**HAL_RTCEx_SetRefClock**](#)
- [**HAL_RTCEx_DeactivateRefClock**](#)
- [**HAL_RTCEx_EnableBypassShadow**](#)
- [**HAL_RTCEx_DisableBypassShadow**](#)

59.2.5 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- [**HAL_RTCEx_AlarmBEventCallback**](#)
- [**HAL_RTCEx_PollForAlarmBEvent**](#)

59.2.6 Detailed description of functions

HAL_RTCEx_SetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
 - **RTC_TIMESTAMPEDGE_RISING:** the Time stamp event occurs on the rising edge of the related pin.
 - **RTC_TIMESTAMPEDGE_FALLING:** the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
 - **RTC_TIMESTAMPPIN_DEFAULT:** PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_SetTimeStamp_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp with Interrupt.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
 - RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
 - RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** Specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
 - RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_DeactivateTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Set Internal TimeStamp.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the internal TimeStamp feature.

HAL_RTCEx_DeactivateInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate Internal TimeStamp.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **HAL:** status

HAL_RTCEx_GetTimeStamp

Function name

```
HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef *  
sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)
```

Function description

Get the RTC TimeStamp value.

Parameters

- **hrtc:** RTC handle
- **sTimeStamp:** Pointer to Time structure
- **sTimeStampDate:** Pointer to Date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
 - RTC_FORMAT_BIN: Binary data format
 - RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTCEx_SetTamper

Function name

```
HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef *  
sTamper)
```

Function description

Set Tamper.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

Notes

- By calling this API we disable the tamper interrupt for all tampers.

HAL_RTCEx_SetTamper_IT

Function name

```
HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef *  
sTamper)
```

Function description

Set Tamper with interrupt.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to RTC Tamper.

Return values

- **HAL:** status

Notes

- By calling this API we force the tamper interrupt for all tampers.

HAL_RTCEx_DeactivateTamper

Function name

```
HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)
```

Function description

Deactivate Tamper.

Parameters

- **hrtc:** RTC handle
- **Tamper:** Selected tamper pin. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3.

Return values

- **HAL:** status

HAL_RTCEx_TamperTimeStampIRQHandler

Function name

```
void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)
```

Function description

Handle TimeStamp interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper1EventCallback

Function name

```
void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)
```

Function description

Tamper 1 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper2EventCallback

Function name

```
void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)
```

Function description

Tamper 2 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper3EventCallback

Function name

```
void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)
```

Function description

Tamper 3 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_TimeStampEventCallback

Function name

```
void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)
```

Function description

TimeStamp callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForTimeStampEvent

Function name

```
HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
```

Function description

Handle TimeStamp polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_PollForTamper1Event

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Tamper1 Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_PollForTamper2Event

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Tamper2 Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_PollForTamper3Event

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Tamper3 Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Set wake up timer.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values

- **HAL:** status

HAL_RTCEx_SetWakeUpTimer_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Set wake up timer with interrupt.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values

- **HAL:** status

HAL_RTCEx_DeactivateWakeUpTimer

Function name

uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Deactivate wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetWakeUpTimer

Function name

uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Get wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **Counter:** value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name

`void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)`

Function description

Handle Wake Up Timer interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_WakeUpTimerEventCallback

Function name

`void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)`

Function description

Wake Up Timer callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForWakeUpTimerEvent

Function name

`HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)`

Function description

Handle Wake Up Timer Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_BKUPWrite

Function name

`void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)`

Function description

Write a data in a specified RTC Backup data register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.

- **Data:** Data to be written in the specified RTC Backup data register.

Return values

- **None:**

HAL_RTCEx_BKUPRead

Function name

```
uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)
```

Function description

Read data from the specified RTC Backup data Register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.

Return values

- **Read:** value

HAL_RTCEx_SetSmoothCalib

Function name

```
HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)
```

Function description

Set the Smooth calibration parameters.

Parameters

- **hrtc:** RTC handle
- **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be one of the following values :
 - RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s.
 - RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s.
 - RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s.
- **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values:
 - RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses.
 - RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.
- **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.

Return values

- **HAL:** status

Notes

- To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

HAL_RTCEx_SetSynchroShift

Function name

```
HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)
```

Function description

Configure the Synchronization Shift Control Settings.

Parameters

- **hrtc:** RTC handle
- **ShiftAdd1S:** Select to add or not 1 second to the time calendar. This parameter can be one of the following values :
 - RTC_SHIFTADD1S_SET: Add one second to the clock calendar.
 - RTC_SHIFTADD1S_RESET: No effect.
- **ShiftSubFS:** Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.

Return values

- **HAL:** status

Notes

- When REFCKON is set, firmware must not write to Shift control register.

HAL_RTCEx_SetCalibrationOutPut

Function name

```
HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)
```

Function description

Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle
- **CalibOutput:** : Select the Calibration output Selection . This parameter can be one of the following values:
 - RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.
 - RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateCalibrationOutPut

Function name

```
HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)
```

Function description

Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetRefClock

Function name

```
HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)
```

Function description

Enable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_DeactivateRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)

Function description

Disable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_EnableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Enable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_DisableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Disable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEEx_AlarmBEventCallback

Function name

void HAL_RTCEEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm B callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEEx_PollForAlarmBEvent

Function name

HAL_StatusTypeDef HAL_RTCEEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Alarm B Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

59.3 RTCEEx Firmware driver defines

59.3.1 RTCEEx

RTC Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTC Backup Registers Definitions

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC_BKP_DR5

RTC_BKP_DR6

RTC_BKP_DR7

RTC_BKP_DR8

RTC_BKP_DR9

RTC_BKP_DR10

RTC_BKP_DR11

RTC_BKP_DR12

RTC_BKP_DR13

RTC_BKP_DR14

RTC_BKP_DR15

RTC_BKP_DR16

RTC_BKP_DR17

RTC_BKP_DR18

RTC_BKP_DR19

RTC_BKP_DR20

RTC_BKP_DR21

RTC_BKP_DR22

RTC_BKP_DR23

RTC_BKP_DR24

RTC_BKP_DR25

RTC_BKP_DR26

RTC_BKP_DR27

RTC_BKP_DR28

RTC_BKP_DR29

RTC_BKP_DR30

RTC_BKP_DR31

RTC Calib Output Selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

RTCEx Exported Macros

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to check. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE](#)**Description:**

- Check whether the specified RTC Wake Up timer interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_GET_FLAG](#)**Description:**

- Get the selected RTC WakeUpTimer's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
 - RTC_FLAG_WUTF
 - RTC_FLAG_WUTWF

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG](#)**Description:**

- Clear the RTC Wake Up timer's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
 - RTC_FLAG_WUTF

Return value:

- None

[__HAL_RTC_TAMPER1_ENABLE](#)**Description:**

- Enable the RTC Tamper1 input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_TAMPER1_DISABLE](#)**Description:**

- Disable the RTC Tamper1 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_TAMPER2_ENABLE](#)**Description:**

- Enable the RTC Tamper2 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_TAMPER2_DISABLE](#)**Description:**

- Disable the RTC Tamper2 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_TAMPER3_ENABLE](#)**Description:**

- Enable the RTC Tamper3 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_TAMPER3_DISABLE](#)**Description:**

- Disable the RTC Tamper3 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_TAMPER_ENABLE_IT](#)**Description:**

- Enable the RTC Tamper interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
 - `RTC_IT_TAMP`: All tampers interrupts
 - `RTC_IT_TAMP1`: Tamper1 interrupt
 - `RTC_IT_TAMP2`: Tamper2 interrupt

- RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

[__HAL_RTC_TAMPER_DISABLE_IT](#)**Description:**

- Disable the RTC Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
 - RTC_IT_TAMP: All tampers interrupts
 - RTC_IT_TAMP1: Tamper1 interrupt
 - RTC_IT_TAMP2: Tamper2 interrupt
 - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

[__HAL_RTC_TAMPER_GET_IT](#)**Description:**

- Check whether the specified RTC Tamper interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
 - RTC_IT_TAMP1: Tamper1 interrupt
 - RTC_IT_TAMP2: Tamper2 interrupt
 - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

[__HAL_RTC_TAMPER_GET_IT_SOURCE](#)**Description:**

- Check whether the specified RTC Tamper interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
 - RTC_IT_TAMP: All tampers interrupts
 - RTC_IT_TAMP1: Tamper1 interrupt
 - RTC_IT_TAMP2: Tamper2 interrupt
 - RTC_IT_TAMP3: Tamper3 interrupt

Return value:

- None

[__HAL_RTC_TAMPER_GET_FLAG](#)**Description:**

- Get the selected RTC Tamper's flag status.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag is pending or not. This parameter can be:
 - `RTC_FLAG_TAMP1F`: Tamper1 flag
 - `RTC_FLAG_TAMP2F`: Tamper2 flag
 - `RTC_FLAG_TAMP3F`: Tamper3 flag

Return value:

- None

[`__HAL_RTC_TAMPER_CLEAR_FLAG`](#)**Description:**

- Clear the RTC Tamper's pending flags.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag sources to clear. This parameter can be:
 - `RTC_FLAG_TAMP1F`: Tamper1 flag
 - `RTC_FLAG_TAMP2F`: Tamper2 flag
 - `RTC_FLAG_TAMP3F`: Tamper3 flag

Return value:

- None

[`__HAL_RTC_TIMESTAMP_ENABLE`](#)**Description:**

- Enable the RTC TimeStamp peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[`__HAL_RTC_TIMESTAMP_DISABLE`](#)**Description:**

- Disable the RTC TimeStamp peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[`__HAL_RTC_TIMESTAMP_ENABLE_IT`](#)**Description:**

- Enable the RTC TimeStamp interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
 - `RTC_IT_TS`: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE_IT

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
 - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to check. This parameter can be:
 - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT_SOURCE

Description:

- Check whether the specified RTC Time Stamp interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
 - RTC_IT_TS: TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC TimeStamp's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
 - RTC_FLAG_TSF
 - RTC_FLAG_TSOVF

Return value:

- None

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamp's pending flags.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to clear. This parameter can be:
 - `RTC_FLAG_TSF`
 - `RTC_FLAG_TSOVF`

Return value:

- None

`__HAL_RTC_INTERNAL_TIMESTAMP_ENABLE`

Description:

- Enable the RTC internal TimeStamp peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_INTERNAL_TIMESTAMP_DISABLE`

Description:

- Disable the RTC internal TimeStamp peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG`

Description:

- Get the selected RTC Internal Time Stamp's flag status.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Internal Time Stamp Flag is pending or not. This parameter can be:
 - `RTC_FLAG_ITSF`

Return value:

- None

`__HAL_RTC_INTERNAL_TIMESTAMP_CLEAR_FLAG`

Description:

- Clear the RTC Internal Time Stamp's pending flags.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Internal Time Stamp Flag source to clear. This parameter can be:
 - `RTC_FLAG_ITSF`

Return value:

- None

`__HAL_RTC_CALIBRATION_OUTPUT_ENABLE`

Description:

- Enable the RTC calibration output.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_CALIBRATION_OUTPUT_DISABLE](#)**Description:**

- Disable the calibration output.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_CLOCKREF_DETECTION_ENABLE](#)**Description:**

- Enable the clock reference detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_CLOCKREF_DETECTION_DISABLE](#)**Description:**

- Disable the clock reference detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

[__HAL_RTC_SHIFT_GET_FLAG](#)**Description:**

- Get the selected RTC shift operation's flag status.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC shift operation Flag is pending or not. This parameter can be:
 - `RTC_FLAG_SHPF`

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_IT](#)**Description:**

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_IT](#)

Description:

- Disable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_EVENT](#)

Description:

- Enable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_EVENT](#)

Description:

- Disable event on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTID3_ENABLE_EVENT](#)

Description:

- Enable event on the RTC WakeUp Timer associated D3 Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTID3_DISABLE_EVENT](#)

Description:

- Disable event on the RTC WakeUp Timer associated D3 Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_FALLING_EDGE](#)

Description:

- Enable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE](#)

Description:

- Disable falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE](#)

Description:

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE](#)

Description:

- Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE](#)

Description:

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE](#)

Description:

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTI_GET_FLAG](#)

Description:

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

[__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG](#)

Description:

- Clear the RTC WakeUp Timer associated Exti line flag.

Return value:

- None

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_IT](#)

Description:

- Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_IT](#)

Description:

- Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_EVENT](#)

Description:

- Enable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_EVENT](#)**Description:**

- Disable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_EXTID3_GET_FLAG](#)**Description:**

- Check whether the RTC WakeUp Timer associated D3 Exti line interrupt flag is set or not.

Return value:

- Line: Status

[__HAL_RTC_WAKEUPTIMER_EXTID3_CLEAR_FLAG](#)**Description:**

- Clear the RTC WakeUp Timer associated D3 Exti line flag.

Return value:

- None.

[__HAL_RTC_WAKEUPTIMER_EXTI_GENERATE_SWIT](#)**Description:**

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_FALLING_EDGE](#)**Description:**

- Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_FALLING_EDGE](#)**Description:**

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE](#)**Description:**

- Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

[__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE](#)**Description:**

- Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

_HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_EDGE**Description:**

- Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

_HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_FALLING_EDGE**Description:**

- Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

_HAL_RTC_TAMPER_TIMESTAMP_EXTI_GET_FLAG**Description:**

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

Return value:

- Line: Status

_HAL_RTC_TAMPER_TIMESTAMP_EXTI_CLEAR_FLAG**Description:**

- Clear the RTC Tamper and Timestamp associated Exti line flag.

Return value:

- None

_HAL_RTC_TAMPER_TIMESTAMP_EXTI_GENERATE_SWIT**Description:**

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

Private macros to check input parameters**IS_RTC_OUTPUT****IS_RTC_BKP****IS_TIMESTAMP_EDGE****IS_RTC_TAMPER****IS_RTC_TAMPER_INTERRUPT****IS_RTC_TIMESTAMP_PIN****IS_RTC_TAMPER_TRIGGER**

IS_RTC_TAMPER_ERASE_MODE
IS_RTC_TAMPER_MASKFLAG_STATE
IS_RTC_TAMPER_FILTER
IS_RTC_TAMPER_SAMPLING_FREQ
IS_RTC_TAMPER_PRECHARGE_DURATION
IS_RTC_TAMPER_TIMESTAMPTAMPER_DETECTION
IS_RTC_TAMPER_PULLUP_STATE
IS_RTC_WAKEUP_CLOCK
IS_RTC_WAKEUP_COUNTER
IS_RTC_SMOOTH_CALIB_PERIOD
IS_RTC_SMOOTH_CALIB_PLUS
IS_RTC_SMOOTH_CALIB_MINUS
IS_RTC_SHIFT_ADD1S
IS_RTC_SHIFT_SUBFS
IS_RTC_CALIB_OUTPUT

RTC Output Selection Definitions

RTC_OUTPUT_DISABLE
RTC_OUTPUT_ALARMA
RTC_OUTPUT_ALARMB
RTC_OUTPUT_WAKEUP

RTC Smooth Calib Period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else $2^{\text{exp}20}$ RTCCLK seconds

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else $2^{\text{exp}19}$ RTCCLK seconds

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else $2^{\text{exp}18}$ RTCCLK seconds

RTC Smooth Calib Plus Pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET

The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLUSPULSES_RESET

The number of RTCCLK pulses substituted during a 32-second window = CALM[8:0]

RTC Tamper EraseBackUp Definitions**RTC_TAMPER_ERASE_BACKUP_ENABLE****RTC_TAMPER_ERASE_BACKUP_DISABLE*****RTC Tamper Filter Definitions*****RTC_TAMPERFILTER_DISABLE**

Tamper filter is disabled

RTC_TAMPERFILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

RTC_TAMPERFILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

RTC_TAMPERFILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level.

RTC Tamper Interrupts Definitions**RTC_TAMPER1_INTERRUPT****RTC_TAMPER2_INTERRUPT****RTC_TAMPER3_INTERRUPT****RTC_ALL_TAMPER_INTERRUPT*****RTC Tamper Mask Flag Definitions*****RTC_TAMPERMASK_FLAG_DISABLE****RTC_TAMPERMASK_FLAG_ENABLE*****RTC Tamper Pins Definitions*****RTC_TAMPER_1****RTC_TAMPER_2****RTC_TAMPER_3*****RTC Tamper Pin Precharge Duration Definitions*****RTC_TAMPERPRECHARGEDURATION_1RTCCLK**

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

RTC_TAMPERPRECHARGEDURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

*RTC Tamper Pull Up Definitions***RTC_TAMPER_PULLUP_ENABLE**

TimeStamp on Tamper Detection event saved

RTC_TAMPER_PULLUP_DISABLE

TimeStamp on Tamper Detection event is not saved

*RTC Tamper Sampling Frequencies Definitions***RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768**

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

*RTC Tamper TimeStamp On Tamper Detection Definitions***RTC_TIMESTAMPONTAMPERDETECTION_ENABLE**

TimeStamp on Tamper Detection event saved

RTC_TIMESTAMPONTAMPERDETECTION_DISABLE

TimeStamp on Tamper Detection event is not saved

*RTC Tamper Triggers Definitions***RTC_TAMPERTRIGGER_RISINGEDGE****RTC_TAMPERTRIGGER_FALLINGEDGE****RTC_TAMPERTRIGGER_LOWLEVEL****RTC_TAMPERTRIGGER_HIGHLEVEL***RTC TimeStamp Edges Definitions***RTC_TIMESTAMPEDGE_RISING****RTC_TIMESTAMPEDGE_FALLING***RTC TimeStamp Pins Selection*

`RTC_TIMESTAMPIN_DEFAULT`

RTC Wakeup Timer Definitions

`RTC_WAKEUPCLOCK_RTCCLK_DIV16`

`RTC_WAKEUPCLOCK_RTCCLK_DIV8`

`RTC_WAKEUPCLOCK_RTCCLK_DIV4`

`RTC_WAKEUPCLOCK_RTCCLK_DIV2`

`RTC_WAKEUPCLOCK_CK_SPRE_16BITS`

`RTC_WAKEUPCLOCK_CK_SPRE_17BITS`

60 HAL SAI Generic Driver

60.1 SAI Firmware driver registers structures

60.1.1 SAI_PdmlInitTypeDef

Data Fields

- *FunctionalState Activation*
- *uint32_t MicPairsNbr*
- *uint32_t ClockEnable*

Field Documentation

- *FunctionalState SAI_PdmlInitTypeDef::Activation*
Enable/Disable PDM interface
- *uint32_t SAI_PdmlInitTypeDef::MicPairsNbr*
Specifies the number of microphone pairs used. This parameter must be a number between Min_Data = 1 and Max_Data = 4.
- *uint32_t SAI_PdmlInitTypeDef::ClockEnable*
Specifies which clock must be enabled. This parameter can be a values combination of **SAI PDM Clock Enable**

60.1.2 SAI_InitTypeDef

Data Fields

- *uint32_t AudioMode*
- *uint32_t Synchro*
- *uint32_t SynchroExt*
- *uint32_t OutputDrive*
- *uint32_t NoDivider*
- *uint32_t FIFOThreshold*
- *uint32_t AudioFrequency*
- *uint32_t Mckdiv*
- *uint32_t MckOverSampling*
- *uint32_t MonoStereoMode*
- *uint32_t CompandingMode*
- *uint32_t TriState*
- *SAI_PdmlInitTypeDef PdmlInit*
- *uint32_t Protocol*
- *uint32_t DataSize*
- *uint32_t FirstBit*
- *uint32_t ClockStrobing*

Field Documentation

- *uint32_t SAI_InitTypeDef::AudioMode*
Specifies the SAI Block audio Mode. This parameter can be a value of **SAI Block Mode**
- *uint32_t SAI_InitTypeDef::Synchro*

Specifies SAI Block synchronization This parameter can be a value of **SAI Block Synchronization**

- **`uint32_t SAI_InitTypeDef::SynchroExt`**

Specifies SAI external output synchronization, this setup is common for BlockA and BlockB This parameter can be a value of **SAI External synchronisation**

Note:

- : If both audio blocks of same SAI are used, this parameter has to be set to the same value for each audio block

- **`uint32_t SAI_InitTypeDef::OutputDrive`**

Specifies when SAI Block outputs are driven. This parameter can be a value of **SAI Block Output Drive**

Note:

- this value has to be set before enabling the audio block but after the audio block configuration.

- **`uint32_t SAI_InitTypeDef::NoDivider`**

Specifies whether master clock will be divided or not. This parameter can be a value of **SAI Block NoDivider**

Note:

- : If bit NOMCK in the SAI_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NOMCK in the SAI_xCR1 register is set, the frame length can take any of the values without constraint since the input clock of the audio block should be equal to the bit clock. There is no MCLK_x clock which can be output.

- **`uint32_t SAI_InitTypeDef::FIFOThreshold`**

Specifies SAI Block FIFO threshold. This parameter can be a value of **SAI Block Fifo Threshold**

- **`uint32_t SAI_InitTypeDef::AudioFrequency`**

Specifies the audio frequency sampling. This parameter can be a value of **SAI Audio Frequency**

- **`uint32_t SAI_InitTypeDef::Mckdiv`**

Specifies the master clock divider, the parameter will be used if for AudioFrequency the user choice This parameter must be a number between Min_Data = 0 and Max_Data = 63

- **`uint32_t SAI_InitTypeDef::MckOverSampling`**

Specifies the master clock oversampling. This parameter can be a value of **SAI Block Master Clock OverSampling**

- **`uint32_t SAI_InitTypeDef::MonoStereoMode`**

Specifies if the mono or stereo mode is selected. This parameter can be a value of **SAI Mono Stereo Mode**

- **`uint32_t SAI_InitTypeDef::CompandingMode`**

Specifies the companding mode type. This parameter can be a value of **SAI Block Companding Mode**

- **`uint32_t SAI_InitTypeDef::TriState`**

Specifies the companding mode type. This parameter can be a value of **SAI TRISState Management**

- **`SAI_PdmInitTypeDef SAI_InitTypeDef::PdmInit`**

Specifies the PDM configuration.

- **`uint32_t SAI_InitTypeDef::Protocol`**

Specifies the SAI Block protocol. This parameter can be a value of **SAI Block Protocol**

- **`uint32_t SAI_InitTypeDef::DataSize`**

Specifies the SAI Block data size. This parameter can be a value of **SAI Block Data Size**

- **`uint32_t SAI_InitTypeDef::FirstBit`**

Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of **SAI Block MSB LSB transmission**

- *uint32_t SAI_InitTypeDef::ClockStrobing*

Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of **SAI Block Clock Strobing**

60.1.3 SAI_FrameInitTypeDef

Data Fields

- *uint32_t FrameLength*
- *uint32_t ActiveFrameLength*
- *uint32_t FSDefinition*
- *uint32_t FSPolarity*
- *uint32_t FSOFFSET*

Field Documentation

- *uint32_t SAI_FrameInitTypeDef::FrameLength*

Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.

Note:

– : If master clock MCLK_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.

- *uint32_t SAI_FrameInitTypeDef::ActiveFrameLength*

Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- *uint32_t SAI_FrameInitTypeDef::FSDefinition*

Specifies the Frame synchronization definition. This parameter can be a value of **SAI Block FS Definition**

- *uint32_t SAI_FrameInitTypeDef::FSPolarity*

Specifies the Frame synchronization Polarity. This parameter can be a value of **SAI Block FS Polarity**

- *uint32_t SAI_FrameInitTypeDef::FSOffset*

Specifies the Frame synchronization Offset. This parameter can be a value of **SAI Block FS Offset**

60.1.4 SAI_SlotInitTypeDef

Data Fields

- *uint32_t FirstBitOffset*
- *uint32_t SlotSize*
- *uint32_t SlotNumber*
- *uint32_t SlotActive*

Field Documentation

- *uint32_t SAI_SlotInitTypeDef::FirstBitOffset*

Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min_Data = 0 and Max_Data = 24

- *uint32_t SAI_SlotInitTypeDef::SlotSize*

Specifies the Slot Size. This parameter can be a value of **SAI Block Slot Size**

- *uint32_t SAI_SlotInitTypeDef::SlotNumber*

Specifies the number of slot in the audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 16

- `uint32_t SAI_SlotInitTypeDef::SlotActive`

Specifies the slots in audio frame that will be activated. This parameter can be a value of **SAI Block Slot Active**

60.1.5 __SAI_HandleTypeDef

Data Fields

- `SAI_Block_TypeDef * Instance`
- `SAI_InitTypeDef Init`
- `SAI_FrameInitTypeDef FrameInit`
- `SAI_SlotInitTypeDef SlotInit`
- `uint8_t * pBuffPtr`
- `uint16_t XferSize`
- `uint16_t XferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `SAIcallback mutecallback`
- `void(* InterruptServiceRoutine`
- `HAL_LockTypeDef Lock`
- `__IO HAL_SAI_StateTypeDef State`
- `__IO uint32_t ErrorCode`

Field Documentation

- **`SAI_Block_TypeDef* __SAI_HandleTypeDef::Instance`**
SAI Blockx registers base address
- **`SAI_InitTypeDef __SAI_HandleTypeDef::Init`**
SAI communication parameters
- **`SAI_FrameInitTypeDef __SAI_HandleTypeDef::FrameInit`**
SAI Frame configuration parameters
- **`SAI_SlotInitTypeDef __SAI_HandleTypeDef::SlotInit`**
SAI Slot configuration parameters
- **`uint8_t* __SAI_HandleTypeDef::pBuffPtr`**
Pointer to SAI transfer Buffer
- **`uint16_t __SAI_HandleTypeDef::XferSize`**
SAI transfer size
- **`uint16_t __SAI_HandleTypeDef::XferCount`**
SAI transfer counter
- **`DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmatx`**
SAI Tx DMA handle parameters
- **`DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmarx`**
SAI Rx DMA handle parameters
- **`SAIcallback __SAI_HandleTypeDef::mutecallback`**
SAI mute callback
- **`void(* __SAI_HandleTypeDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDef *hsai)`**
- **`HAL_LockTypeDef __SAI_HandleTypeDef::Lock`**

- SAI locking object
- `_IO HAL_SAI_StateTypeDef __SAI_HandleTypeDefDef::State`
 - SAI communication state
- `_IO uint32_t __SAI_HandleTypeDefDef::ErrorCode`
 - SAI Error code

60.2 SAI Firmware driver API description

60.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a SAI_HandleTypeDef handle structure (eg. SAI_HandleTypeDef hsai).
2. Initialize the SAI low level resources by implementing the HAL_SAI_MspInit() API:
 - a. Enable the SAI interface clock.
 - b. SAI pins configuration:
 - Enable the clock for the SAI GPIOs.
 - Configure these SAI pins as alternate function pull-up.
 - c. NVIC configuration if you need to use interrupt process (HAL_SAI_Transmit_IT() and HAL_SAI_Receive_IT() APIs):
 - Configure the SAI interrupt priority.
 - Enable the NVIC SAI IRQ handle.
 - d. DMA Configuration if you need to use DMA process (HAL_SAI_Transmit_DMA()) and HAL_SAI_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx stream.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx Stream.
 - Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. The initialization can be done by two ways
 - a. Expert mode : Initialize the structures Init, FrameInit and SlotInit and call HAL_SAI_Init().
 - b. Simplified mode : Initialize the high part of Init Structure and call HAL_SAI_InitProtocol().

Note: *The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros `_HAL_SAI_ENABLE_IT()` and `_HAL_SAI_DISABLE_IT()` inside the transmit and receive process.*

Note: *Make sure that either:*

- PLLSAI1CLK output is configured or
- PLLSAI2CLK output is configured or
- PLLSAI3CLK output is configured or
- PLLSAI4ACLK output is configured or
- PLLSAI4BCLK output is configured or
- External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the `stm32h7xx_hal_conf.h` file.

Note: *In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.*

Note: *In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.*

Note: *It is mandatory to respect the following conditions in order to avoid bad SAI behavior:*

- *First bit Offset <= (SLOT size - Data size)*
- *Data size <= SLOT size*
- *Number of SLOT x SLOT size = Frame length*
- *The number of slots should be even when SAI_FS_CHANNEL_IDENTIFICATION is selected.*

Note: PDM interface can be activated through HAL_SAI_Init function. Please note that PDM interface is only available for SAIx sub-block A. PDM microphone delays can be tuned with HAL_SAIEx_ConfigPdmMicDelay function.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SAI_Transmit()
- Receive an amount of data in blocking mode using HAL_SAI_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SAI_Transmit_IT()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SAI_Receive_IT()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SAI_Transmit_DMA()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SAI_Receive_DMA()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()
- Pause the DMA Transfer using HAL_SAI_DMAPause()
- Resume the DMA Transfer using HAL_SAI_DMAResume()
- Stop the DMA Transfer using HAL_SAI_DMAStop()

SAI HAL driver additional function list

Below the list the others API available SAI HAL driver :

- HAL_SAI_EnableTxMuteMode(): Enable the mute in tx mode
- HAL_SAI_DisableTxMuteMode(): Disable the mute in tx mode
- HAL_SAI_EnableRxMuteMode(): Enable the mute in Rx mode
- HAL_SAI_DisableRxMuteMode(): Disable the mute in Rx mode
- HAL_SAI_FlushRxFifo(): Flush the rx fifo.
- HAL_SAI_Abort(): Abort the current transfer

SAI HAL driver macros list

Below the list of most used macros in SAI HAL driver :

- __HAL_SAI_ENABLE(): Enable the SAI peripheral
- __HAL_SAI_DISABLE(): Disable the SAI peripheral
- __HAL_SAI_ENABLE_IT(): Enable the specified SAI interrupts

- `__HAL_SAI_DISABLE_IT()`: Disable the specified SAI interrupts
- `__HAL_SAI_GET_IT_SOURCE()`: Check if the specified SAI interrupt source is enabled or disabled
- `__HAL_SAI_GET_FLAG()`: Check whether the specified SAI flag is set or not

60.2.2

Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SAIx peripheral:

- User must implement `HAL_SAI_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function `HAL_SAI_Init()` to configure the selected device with the selected configuration:
 - Mode (Master/slave TX/RX)
 - Protocol
 - Data Size
 - MCLK Output
 - Audio frequency
 - FIFO Threshold
 - Frame Config
 - Slot Config
 - PDM Config
- Call the function `HAL_SAI_DeInit()` to restore the default configuration of the selected SAI peripheral.

This section contains the following APIs:

- [`HAL_SAI_InitProtocol`](#)
- [`HAL_SAI_Init`](#)
- [`HAL_SAI_DeInit`](#)
- [`HAL_SAI_MspInit`](#)
- [`HAL_SAI_MspDeInit`](#)

60.2.3

IO operation functions

This subsection provides a set of functions allowing to manage the SAI data transfers.

- There are two modes of transfer:
 - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SAI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- Blocking mode functions are :
 - `HAL_SAI_Transmit()`
 - `HAL_SAI_Receive()`
 - `HAL_SAI_TransmitReceive()`
- Non Blocking mode functions with Interrupt are :
 - `HAL_SAI_Transmit_IT()`
 - `HAL_SAI_Receive_IT()`
 - `HAL_SAI_TransmitReceive_IT()`
- Non Blocking mode functions with DMA are :
 - `HAL_SAI_Transmit_DMA()`
 - `HAL_SAI_Receive_DMA()`
 - `HAL_SAI_TransmitReceive_DMA()`
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - `HAL_SAI_TxCpltCallback()`

- HAL_SAI_RxCpltCallback()
- HAL_SAI_ErrorCallback()

This section contains the following APIs:

- [HAL_SAI_Transmit](#)
- [HAL_SAI_Receive](#)
- [HAL_SAI_Transmit_IT](#)
- [HAL_SAI_Receive_IT](#)
- [HAL_SAI_DMAPause](#)
- [HAL_SAI_DMAResume](#)
- [HAL_SAI_DMAStop](#)
- [HAL_SAI_Abort](#)
- [HAL_SAI_Transmit_DMA](#)
- [HAL_SAI_Receive_DMA](#)
- [HAL_SAI_EnableTxMuteMode](#)
- [HAL_SAI_DisableTxMuteMode](#)
- [HAL_SAI_EnableRxMuteMode](#)
- [HAL_SAI_DisableRxMuteMode](#)
- [HAL_SAI_IRQHandler](#)
- [HAL_SAI_TxCpltCallback](#)
- [HAL_SAI_TxHalfCpltCallback](#)
- [HAL_SAI_RxCpltCallback](#)
- [HAL_SAI_RxHalfCpltCallback](#)
- [HAL_SAI_ErrorCallback](#)

60.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- [HAL_SAI_GetState](#)
- [HAL_SAI_GetError](#)

60.2.5 Detailed description of functions

HAL_SAI_InitProtocol

Function name

HAL_StatusTypeDef HAL_SAI_InitProtocol (SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)

Function description

Initialize the structure FrameInit, SlotInit and the low part of Init according to the specified parameters and call the function HAL_SAI_Init to initialize the SAI block.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **protocol:** one of the supported protocol SAI Supported protocol
- **datasize:** one of the supported datasize SAI protocol data size the configuration information for SAI module.
- **nbslot:** Number of slot.

Return values

- **HAL:** status

HAL_SAI_Init**Function name**

HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI according to the specified parameters.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DelInit**Function name**

HAL_StatusTypeDef HAL_SAI_DelInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI peripheral.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_MspInit**Function name**

void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_MspDelInit**Function name**

void HAL_SAI_MspDelInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_Transmit

Function name

```
HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Receive

Function name

```
HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Transmit_IT

Function name

```
HAL_StatusTypeDef HAL_SAI_Transmit_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)
```

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_IT

Function name

`HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)`

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_Transmit_DMA

Function name

`HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)`

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_DMA

Function name

`HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)`

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_DMAPause

Function name

`HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)`

Function description

Pause the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAResume**Function name**

HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)

Function description

Resume the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAStop**Function name**

HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)

Function description

Stop the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_Abort**Function name**

HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)

Function description

Abort the current transfer and disable the SAI.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableTxMuteMode**Function name**

HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)

Function description

Enable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **val:** value sent during the mute SAI Block Mute Value

Return values

- **HAL:** status

HAL_SAI_DisableTxMuteMode

Function name

```
HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)
```

Function description

Disable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableRxMuteMode

Function name

```
HAL_StatusTypeDef HAL_SAI_EnableRxMuteMode (SAI_HandleTypeDef * hsai, SAIcallback callback,  
uint16_t counter)
```

Function description

Enable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **callback:** function called when the mute is detected.
- **counter:** number a data before mute detection max 63.

Return values

- **HAL:** status

HAL_SAI_DisableRxMuteMode

Function name

```
HAL_StatusTypeDef HAL_SAI_DisableRxMuteMode (SAI_HandleTypeDef * hsai)
```

Function description

Disable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_IRQHandler

Function name

```
void HAL_SAI_IRQHandler (SAI_HandleTypeDef * hsai)
```

Function description

Handle SAI interrupt request.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxHalfCpltCallback

Function name

void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer Half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxCpltCallback

Function name

void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxHalfCpltCallback

Function name

void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxCpltCallback

Function name

void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_ErrorCallback

Function name

```
void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)
```

Function description

SAI error callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_GetState

Function name

```
HAL_SAI_StateTypeDef HAL_SAI_GetState (SAI_HandleTypeDef * hsai)
```

Function description

Return the SAI handle state.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** state

HAL_SAI_GetError

Function name

```
uint32_t HAL_SAI_GetError (SAI_HandleTypeDef * hsai)
```

Function description

Return the SAI error code.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for the specified SAI Block.

Return values

- **SAI:** Error Code

60.3 SAI Firmware driver defines

60.3.1 SAI

SAI Audio Frequency

SAI_AUDIO_FREQUENCY_192K

SAI_AUDIO_FREQUENCY_96K

SAI_AUDIO_FREQUENCY_48K

SAI_AUDIO_FREQUENCY_44K

SAI_AUDIO_FREQUENCY_32K

SAI_AUDIO_FREQUENCY_22K

SAI_AUDIO_FREQUENCY_16K

SAI_AUDIO_FREQUENCY_11K

SAI_AUDIO_FREQUENCY_8K

SAI_AUDIO_FREQUENCY_MCKDIV

SAI Block Clock Strobing

SAI_CLOCKSTROBING_FALLINGEDGE

SAI_CLOCKSTROBING_RISINGEDGE

SAI Block Companding Mode

SAI_NOCOMPANDING

SAI_ULAW_1CPL_COMPANDING

SAI_ALAW_1CPL_COMPANDING

SAI_ULAW_2CPL_COMPANDING

SAI_ALAW_2CPL_COMPANDING

SAI Block Data Size

SAI_DATASIZE_8

SAI_DATASIZE_10

SAI_DATASIZE_16

SAI_DATASIZE_20

SAI_DATASIZE_24

SAI_DATASIZE_32

SAI Block Fifo Status Level

SAI_FIFOSTATUS_EMPTY

`SAI_FIFOSTATUS_LESS1QUARTERFULL`

`SAI_FIFOSTATUS_1QUARTERFULL`

`SAI_FIFOSTATUS_HALFFULL`

`SAI_FIFOSTATUS_3QUARTERFULL`

`SAI_FIFOSTATUS_FULL`

SAI Block Fifo Threshold

`SAI_FIFOTHRESHOLD_EMPTY`

`SAI_FIFOTHRESHOLD_1QF`

`SAI_FIFOTHRESHOLD_HF`

`SAI_FIFOTHRESHOLD_3QF`

`SAI_FIFOTHRESHOLD_FULL`

SAI Block Flags Definition

`SAI_FLAG_OVRUDR`

`SAI_FLAG_MUTEDET`

`SAI_FLAG_WCKCFG`

`SAI_FLAG_FREQ`

`SAI_FLAG_CNRDY`

`SAI_FLAG_AFSDET`

`SAI_FLAG_LFSDET`

SAI Block FS Definition

`SAI_FS_STARTFRAME`

`SAI_FS_CHANNEL_IDENTIFICATION`

SAI Block FS Offset

`SAI_FS_FIRSTBIT`

`SAI_FS_BEFOREFIRSTBIT`

SAI Block FS Polarity

`SAI_FS_ACTIVE_LOW`

`SAI_FS_ACTIVE_HIGH`

SAI Block Interrupts Definition

`SAI_IT_OVRUDR`

`SAI_IT_MUTEDET`

`SAI_IT_WCKCFG`

`SAI_IT_FREQ`

`SAI_IT_CNRDY`

`SAI_IT_AFSDET`

`SAI_IT_LFSDET`

SAI Block Master Clock OverSampling

`SAI_MCK_OVERSAMPLING_DISABLE`

`SAI_MCK_OVERSAMPLING_ENABLE`

SAI Block Mode

`SAI_MODEMASTER_TX`

`SAI_MODEMASTER_RX`

`SAI_MODESLAVE_TX`

`SAI_MODESLAVE_RX`

SAI Block MSB LSB transmission

`SAI_FIRSTBIT_MSB`

`SAI_FIRSTBIT_LSB`

SAI Block Mute Value

`SAI_ZERO_VALUE`

`SAI_LAST_SENT_VALUE`

SAI Block NoDivider

`SAI_MASTERDIVIDER_ENABLE`

`SAI_MASTERDIVIDER_DISABLE`

SAI Block Output Drive

`SAI_OUTPUTDRIVE_DISABLE`

`SAI_OUTPUTDRIVE_ENABLE`

SAI Block Protocol

`SAI_FREE_PROTOCOL`

`SAI_SPDIF_PROTOCOL`

`SAI_AC97_PROTOCOL`

SAI Block Slot Active

SAI_SLOT_NOTACTIVE

SAI_SLOTACTIVE_0

SAI_SLOTACTIVE_1

SAI_SLOTACTIVE_2

SAI_SLOTACTIVE_3

SAI_SLOTACTIVE_4

SAI_SLOTACTIVE_5

SAI_SLOTACTIVE_6

SAI_SLOTACTIVE_7

SAI_SLOTACTIVE_8

SAI_SLOTACTIVE_9

SAI_SLOTACTIVE_10

SAI_SLOTACTIVE_11

SAI_SLOTACTIVE_12

SAI_SLOTACTIVE_13

SAI_SLOTACTIVE_14

SAI_SLOTACTIVE_15

SAI_SLOTACTIVE_ALL

SAI Block Slot Size

SAI_SLOTSIZE_DATASIZE

SAI_SLOTSIZE_16B

SAI_SLOTSIZE_32B

SAI External synchronisation

SAI_SYNCEXT_DISABLE

SAI_SYNCEXT_OUTBLOCKA_ENABLE

SAI_SYNCEXT_OUTBLOCKB_ENABLE

SAI Block Synchronization

SAI_ASYNCNHRONOUS

Asynchronous

SAI_SYNCHRONOUS

Synchronous with other block of same SAI

SAI_SYNCHRONOUS_EXT_SAI1

Synchronous with other SAI, SAI1

SAI_SYNCHRONOUS_EXT_SAI2

Synchronous with other SAI, SAI2

SAI Error Code**HAL_SAI_ERROR_NONE**

No error

HAL_SAI_ERROR_OVR

Overrun Error

HAL_SAI_ERROR_UDR

Underrun error

HAL_SAI_ERROR_AFSDET

Anticipated Frame synchronisation detection

HAL_SAI_ERROR_LFSDET

Late Frame synchronisation detection

HAL_SAI_ERROR_CNREADY

codec not ready

HAL_SAI_ERROR_WCKCFG

Wrong clock configuration

HAL_SAI_ERROR_TIMEOUT

Timeout error

HAL_SAI_ERROR_DMA

DMA error

SAI Exported Macros**__HAL_SAI_RESET_HANDLE_STATE****Description:**

- Reset SAI handle state.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_ENABLE_IT**Description:**

- Enable or disable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.

- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - SAI_IT_OVRUDR: Overrun underrun interrupt enable
 - SAI_IT_MUTEDET: Mute detection interrupt enable
 - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
 - SAI_IT_FREQ: FIFO request interrupt enable
 - SAI_IT_CNRDY: Codec not ready interrupt enable
 - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
 - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

[__HAL_SAI_DISABLE_IT](#)[__HAL_SAI_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the SAI interrupt source to check. This parameter can be one of the following values:
 - SAI_IT_OVRUDR: Overrun underrun interrupt enable
 - SAI_IT_MUTEDET: Mute detection interrupt enable
 - SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
 - SAI_IT_FREQ: FIFO request interrupt enable
 - SAI_IT_CNRDY: Codec not ready interrupt enable
 - SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
 - SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

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

[__HAL_SAI_GET_FLAG](#)**Description:**

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

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - SAI_FLAG_OVRUDR: Overrun underrun flag.
 - SAI_FLAG_MUTEDET: Mute detection flag.
 - SAI_FLAG_WCKCFG: Wrong Clock Configuration flag.
 - SAI_FLAG_FREQ: FIFO request flag.
 - SAI_FLAG_CNRDY: Codec not ready flag.
 - SAI_FLAG_AFSDET: Anticipated frame synchronization detection flag.
 - SAI_FLAG_LFSDET: Late frame synchronization detection flag.

Return value:

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

__HAL_SAI_CLEAR_FLAG

Description:

- Clear the specified SAI pending flag.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
 - SAI_FLAG_OVRUDR: Clear Overrun underrun
 - SAI_FLAG_MUTEDET: Clear Mute detection
 - SAI_FLAG_WCKCFG: Clear Wrong Clock Configuration
 - SAI_FLAG_FREQ: Clear FIFO request
 - SAI_FLAG_CNRDY: Clear Codec not ready
 - SAI_FLAG_AFSDET: Clear Anticipated frame synchronization detection
 - SAI_FLAG_LFSDET: Clear Late frame synchronization detection

Return value:

- None

__HAL_SAI_ENABLE

__HAL_SAI_DISABLE

SAI Mono Stereo Mode

SAI_STEREO MODE

SAI_MONOMODE

SAI PDM Clock Enable

SAI_PDM_CLOCK1_ENABLE

SAI_PDM_CLOCK2_ENABLE

SAI_PDM_CLOCK3_ENABLE

SAI_PDM_CLOCK4_ENABLE

SAI Supported protocol

SAI_I2S_STANDARD

SAI_I2S_MSBJUSTIFIED

SAI_I2S_LSBJUSTIFIED

SAI_PCM_LONG

SAI_PCM_SHORT

SAI protocol data size

SAI_PROTOCOL_DATASIZE_16BIT

SAI_PROTOCOL_DATASIZE_16BITEXTENDED

SAI_PROTOCOL_DATASIZE_24BIT

SAI_PROTOCOL_DATASIZE_32BIT

SAI TRIS State Management

SAI_OUTPUT_NOTRELEASED

SAI_OUTPUT_RELEASED

61 HAL SAI Extension Driver

61.1 SAIEx Firmware driver registers structures

61.1.1 SAIEx_PdmMicDelayParamTypeDef

Data Fields

- *uint32_t MicPair*
- *uint32_t LeftDelay*
- *uint32_t RightDelay*

Field Documentation

- *uint32_t SAIEx_PdmMicDelayParamTypeDef::MicPair*

Specifies which pair of microphones is selected. This parameter must be a number between Min_Data = 1 and Max_Data = 4.

- *uint32_t SAIEx_PdmMicDelayParamTypeDef::LeftDelay*

Specifies the delay in PDM clock unit to apply on left microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.

- *uint32_t SAIEx_PdmMicDelayParamTypeDef::RightDelay*

Specifies the delay in PDM clock unit to apply on right microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.

61.2 SAIEx Firmware driver API description

61.2.1 Extended features functions

This section provides functions allowing to:

- Modify PDM microphone delays

This section contains the following APIs:

- [HAL_SAIEx_ConfigPdmMicDelay](#)

61.2.2 Detailed description of functions

[HAL_SAIEx_ConfigPdmMicDelay](#)

Function name

```
HAL_StatusTypeDef HAL_SAIEx_ConfigPdmMicDelay (SAI_HandleTypeDef * hsai,  
SAIEx_PdmMicDelayParamTypeDef * pdmMicDelay)
```

Function description

Configure PDM microphone delays.

Parameters

- **hsai:** SAI handle.
- **pdmMicDelay:** Microphone delays configuration.

Return values

- **HAL:** status

62 HAL SDRAM Generic Driver

62.1 SDRAM Firmware driver registers structures

62.1.1 SDRAM_HandleTypeDef

Data Fields

- *FMC_SDRAM_TypeDef * Instance*
- *FMC_SDRAM_InitTypeDef Init*
- *_IO HAL_SDRAM_StateTypeDef State*
- *HAL_LockTypeDef Lock*
- *MDMA_HandleTypeDef * hmdma*

Field Documentation

- ***FMC_SDRAM_TypeDef* SDRAM_HandleTypeDef::Instance***
Register base address
- ***FMC_SDRAM_InitTypeDef SDRAM_HandleTypeDef::Init***
SDRAM device configuration parameters
- ***_IO HAL_SDRAM_StateTypeDef SDRAM_HandleTypeDef::State***
SDRAM access state
- ***HAL_LockTypeDef SDRAM_HandleTypeDef::Lock***
SDRAM locking object
- ***MDMA_HandleTypeDef* SDRAM_HandleTypeDef::hmdma***
Pointer MDMA handler

62.2 SDRAM Firmware driver API description

62.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SDRAM memories. It uses the FMC layer functions to interface with SDRAM devices. The following sequence should be followed to configure the FMC to interface with SDRAM memories:

1. Declare a SDRAM_HandleTypeDef handle structure, for example: SDRAM_HandleTypeDef hdsram
 - Fill the SDRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
 - Fill the SDRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SDRAM device
2. Declare a FMC_SDRAM_TimingTypeDef structure; for example: FMC_SDRAM_TimingTypeDef Timing; and fill its fields with the allowed values of the structure member.
3. Initialize the SDRAM Controller by calling the function HAL_SDRAM_Init(). This function performs the following sequence:
 - a. MSP hardware layer configuration using the function HAL_SDRAM_MsplInit()
 - b. Control register configuration using the FMC SDRAM interface function FMC_SDRAM_Init()
 - c. Timing register configuration using the FMC SDRAM interface function FMC_SDRAM_Timing_Init()
 - d. Program the SDRAM external device by applying its initialization sequence according to the device plugged in your hardware. This step is mandatory for accessing the SDRAM device.

4. At this stage you can perform read/write accesses from/to the memory connected to the SDRAM Bank. You can perform either polling or DMA transfer using the following APIs:
 - HAL_SDRAM_Read()/HAL_SDRAM_Write() for polling read/write access
 - HAL_SDRAM_Read_DMA()/HAL_SDRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SDRAM device by calling the control APIs HAL_SDRAM_WriteOperation_Enable() / HAL_SDRAM_WriteOperation_Disable() to respectively enable/disable the SDRAM write operation or the function HAL_SDRAM_SendCommand() to send a specified command to the SDRAM device. The command to be sent must be configured with the FMC_SDRAM_CommandTypeDef structure.
6. You can continuously monitor the SDRAM device HAL state by calling the function HAL_SDRAM_GetState()

62.2.2

SDRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SDRAM memory

This section contains the following APIs:

- [**HAL_SDRAM_Init**](#)
- [**HAL_SDRAM_DelInit**](#)
- [**HAL_SDRAM_MsplInit**](#)
- [**HAL_SDRAM_MspDelInit**](#)
- [**HAL_SDRAM_IRQHandler**](#)
- [**HAL_SDRAM_RefreshErrorCallback**](#)
- [**HAL_SDRAM_DMA_XferCpltCallback**](#)
- [**HAL_SDRAM_DMA_XferErrorCallback**](#)

62.2.3

SDRAM Input and Output functions

This section provides functions allowing to use and control the SDRAM memory

This section contains the following APIs:

- [**HAL_SDRAM_Read_8b**](#)
- [**HAL_SDRAM_Write_8b**](#)
- [**HAL_SDRAM_Read_16b**](#)
- [**HAL_SDRAM_Write_16b**](#)
- [**HAL_SDRAM_Read_32b**](#)
- [**HAL_SDRAM_Write_32b**](#)
- [**HAL_SDRAM_Read_DMA**](#)
- [**HAL_SDRAM_Write_DMA**](#)

62.2.4

SDRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SDRAM interface.

This section contains the following APIs:

- [**HAL_SDRAM_WriteProtection_Enable**](#)
- [**HAL_SDRAM_WriteProtection_Disable**](#)
- [**HAL_SDRAM_SendCommand**](#)
- [**HAL_SDRAM_ProgramRefreshRate**](#)
- [**HAL_SDRAM_SetAutoRefreshNumber**](#)
- [**HAL_SDRAM_GetModeStatus**](#)

62.2.5

SDRAM State functions

This subsection permits to get in run-time the status of the SDRAM controller and the data flow.

This section contains the following APIs:

- [*HAL_SDRAM_GetState*](#)

62.2.6 Detailed description of functions

[**HAL_SDRAM_Init**](#)

Function name

HAL_StatusTypeDef HAL_SDRAM_Init (SDRAM_HandleTypeDef * hsdrdram, FMC_SDRAM_TimingTypeDef * Timing)

Function description

Performs the SDRAM device initialization sequence.

Parameters

- **hsdrdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Timing:** Pointer to SDRAM control timing structure

Return values

- **HAL:** status

[**HAL_SDRAM_DelInit**](#)

Function name

HAL_StatusTypeDef HAL_SDRAM_DelInit (SDRAM_HandleTypeDef * hsdrdram)

Function description

Perform the SDRAM device initialization sequence.

Parameters

- **hsdrdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

[**HAL_SDRAM_MspInit**](#)

Function name

void HAL_SDRAM_MspInit (SDRAM_HandleTypeDef * hsdrdram)

Function description

SDRAM MSP Init.

Parameters

- **hsdrdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

[**HAL_SDRAM_MspDelInit**](#)

Function name

void HAL_SDRAM_MspDelInit (SDRAM_HandleTypeDef * hsdrdram)

Function description

SDRAM MSP Delnit.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

HAL_SDRAM_IRQHandler

Function name

void HAL_SDRAM_IRQHandler (SDRAM_HandleTypeDef * hsdram)

Function description

This function handles SDRAM refresh error interrupt request.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_RefreshErrorCallback

Function name

void HAL_SDRAM_RefreshErrorCallback (SDRAM_HandleTypeDef * hsdram)

Function description

SDRAM Refresh error callback.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **None:**

HAL_SDRAM_DMA_XferCpltCallback

Function name

void HAL_SDRAM_DMA_XferCpltCallback (MDMA_HandleTypeDef * hmdma)

Function description

DMA transfer complete callback.

Parameters

- **hmdma:** pointer to a MDMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

HAL_SDRAM_DMA_XferErrorCallback

Function name

```
void HAL_SDRAM_DMA_XferErrorCallback (MDMA_HandleTypeDef * hmdma)
```

Function description

DMA transfer complete error callback.

Parameters

- **hmdma:** DMA handle

Return values

- **None:**

HAL_SDRAM_Read_8b

Function name

```
HAL_StatusTypeDef HAL_SDRAM_Read_8b (SDRAM_HandleTypeDef * hsdr, uint32_t * pAddress,  
uint8_t * pDstBuffer, uint32_t BufferSize)
```

Function description

Reads 8-bit data buffer from the SDRAM memory.

Parameters

- **hsdr:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_8b

Function name

```
HAL_StatusTypeDef HAL_SDRAM_Write_8b (SDRAM_HandleTypeDef * hsdr, uint32_t * pAddress,  
uint8_t * pSrcBuffer, uint32_t BufferSize)
```

Function description

Writes 8-bit data buffer to SDRAM memory.

Parameters

- **hsdr:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_16b

Function name

```
HAL_StatusTypeDef HAL_SDRAM_Read_16b (SDRAM_HandleTypeDef * hsdrdram, uint32_t * pAddress,  
uint16_t * pDstBuffer, uint32_t BufferSize)
```

Function description

Reads 16-bit data buffer from the SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_16b

Function name

```
HAL_StatusTypeDef HAL_SDRAM_Write_16b (SDRAM_HandleTypeDef * hsdrdram, uint32_t * pAddress,  
uint16_t * pSrcBuffer, uint32_t BufferSize)
```

Function description

Writes 16-bit data buffer to SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_32b

Function name

```
HAL_StatusTypeDef HAL_SDRAM_Read_32b (SDRAM_HandleTypeDef * hsdrdram, uint32_t * pAddress,  
uint32_t * pDstBuffer, uint32_t BufferSize)
```

Function description

Reads 32-bit data buffer from the SDRAM memory.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_32b (SDRAM_HandleTypeDef * hsdrdram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 32-bit data buffer to SDRAM memory.

Parameters

- **hsdrdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SDRAM_Read_DMA (SDRAM_HandleTypeDef * hsdrdram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads a Words data from the SDRAM memory using DMA transfer.

Parameters

- **hsdrdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SDRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SDRAM_Write_DMA (SDRAM_HandleTypeDef * hsdrdram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes a Words data buffer to SDRAM memory using DMA transfer.

Parameters

- **hsdrdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SDRAM_WriteProtection_Enable

Function name

HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Enable (SDRAM_HandleTypeDef * hsdr)

Function description

Enables dynamically SDRAM write protection.

Parameters

- **hsdr:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_WriteProtection_Disable

Function name

HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Disable (SDRAM_HandleTypeDef * hsdr)

Function description

Disables dynamically SDRAM write protection.

Parameters

- **hsdr:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** status

HAL_SDRAM_SendCommand

Function name

HAL_StatusTypeDef HAL_SDRAM_SendCommand (SDRAM_HandleTypeDef * hsdr, FMC_SDRAM_CommandTypeDef * Command, uint32_t Timeout)

Function description

Sends Command to the SDRAM bank.

Parameters

- **hsdr:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Command:** SDRAM command structure
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SDRAM_ProgramRefreshRate

Function name

HAL_StatusTypeDef HAL_SDRAM_ProgramRefreshRate (SDRAM_HandleTypeDef * hsdr, uint32_t RefreshRate)

Function description

Programs the SDRAM Memory Refresh rate.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **RefreshRate:** The SDRAM refresh rate value

Return values

- **HAL:** status

HAL_SDRAM_SetAutoRefreshNumber

Function name

HAL_StatusTypeDef HAL_SDRAM_SetAutoRefreshNumber (SDRAM_HandleTypeDef * hsdram, uint32_t AutoRefreshNumber)

Function description

Sets the Number of consecutive SDRAM Memory auto Refresh commands.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **AutoRefreshNumber:** The SDRAM auto Refresh number

Return values

- **HAL:** status

HAL_SDRAM_GetModeStatus

Function name

uint32_t HAL_SDRAM_GetModeStatus (SDRAM_HandleTypeDef * hsdram)

Function description

Returns the SDRAM memory current mode.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **The:** SDRAM memory mode.

HAL_SDRAM_GetState

Function name

HAL_SDRAM_StateTypeDef HAL_SDRAM_GetState (SDRAM_HandleTypeDef * hsdram)

Function description

Returns the SDRAM state.

Parameters

- **hsdram:** pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- **HAL:** state

62.3 SDRAM Firmware driver defines

62.3.1 SDRAM

SDRAM Exported Macros

[__HAL_SDRAM_RESET_HANDLE_STATE](#)

Description:

- Reset SDRAM handle state.

Parameters:

- [__HANDLE__](#): specifies the SDRAM handle.

Return value:

- None

63 HAL SD Generic Driver

63.1 SD Firmware driver registers structures

63.1.1 HAL_SD_CardInfoTypeDef

Data Fields

- `uint32_t CardType`
- `uint32_t CardVersion`
- `uint32_t Class`
- `uint32_t RelCardAdd`
- `uint32_t BlockNbr`
- `uint32_t BlockSize`
- `uint32_t LogBlockNbr`
- `uint32_t LogBlockSize`
- `uint32_t CardSpeed`

Field Documentation

- `uint32_t HAL_SD_CardInfoTypeDef::CardType`
Specifies the card Type
- `uint32_t HAL_SD_CardInfoTypeDef::CardVersion`
Specifies the card version
- `uint32_t HAL_SD_CardInfoTypeDef::Class`
Specifies the class of the card class
- `uint32_t HAL_SD_CardInfoTypeDef::RelCardAdd`
Specifies the Relative Card Address
- `uint32_t HAL_SD_CardInfoTypeDef::BlockNbr`
Specifies the Card Capacity in blocks
- `uint32_t HAL_SD_CardInfoTypeDef::BlockSize`
Specifies one block size in bytes
- `uint32_t HAL_SD_CardInfoTypeDef::LogBlockNbr`
Specifies the Card logical Capacity in blocks
- `uint32_t HAL_SD_CardInfoTypeDef::LogBlockSize`
Specifies logical block size in bytes
- `uint32_t HAL_SD_CardInfoTypeDef::CardSpeed`
Specifies the card Speed

63.1.2 SD_HandleTypeDefDef

Data Fields

- `SD_TypeDef * Instance`
- `SD_InitTypeDef Init`
- `HAL_LockTypeDef Lock`

- `uint32_t * pTxBuffPtr`
- `uint32_t TxXferSize`
- `uint32_t * pRxBuffPtr`
- `uint32_t RxXferSize`
- `__IO uint32_t Context`
- `__IO HAL_SD_StateTypeDef State`
- `__IO uint32_t ErrorCode`
- `HAL_SD_CardInfoTypeDef SdCard`
- `uint32_t CSD`
- `uint32_t CID`

Field Documentation

- `SD_TypeDef* SD_HandleTypeDef::Instance`
SD registers base address
- `SD_InitTypeDef SD_HandleTypeDef::Init`
SD required parameters
- `HAL_LockTypeDef SD_HandleTypeDef::Lock`
SD locking object
- `uint32_t* SD_HandleTypeDef::pTxBuffPtr`
Pointer to SD Tx transfer Buffer
- `uint32_t SD_HandleTypeDef::TxXferSize`
SD Tx Transfer size
- `uint32_t* SD_HandleTypeDef::pRxBuffPtr`
Pointer to SD Rx transfer Buffer
- `uint32_t SD_HandleTypeDef::RxXferSize`
SD Rx Transfer size
- `__IO uint32_t SD_HandleTypeDef::Context`
SD transfer context
- `__IO HAL_SD_StateTypeDef SD_HandleTypeDef::State`
SD card State
- `__IO uint32_t SD_HandleTypeDef::ErrorCode`
SD Card Error codes
- `HAL_SD_CardInfoTypeDef SD_HandleTypeDef::SdCard`
SD Card information
- `uint32_t SD_HandleTypeDef::CSD[4]`
SD card specific data table
- `uint32_t SD_HandleTypeDef::CID[4]`
SD card identification number table

63.1.3 HAL_SD_CardCSDTypeDef

Data Fields

- `__IO uint8_t CSDStruct`
- `__IO uint8_t SysSpecVersion`

- `__IO uint8_t Reserved1`
- `__IO uint8_t TAAC`
- `__IO uint8_t NSAC`
- `__IO uint8_t MaxBusClkFrec`
- `__IO uint16_t CardComdClasses`
- `__IO uint8_t RdBlockLen`
- `__IO uint8_t PartBlockRead`
- `__IO uint8_t WrBlockMisalign`
- `__IO uint8_t RdBlockMisalign`
- `__IO uint8_t DSRImpl`
- `__IO uint8_t Reserved2`
- `__IO uint32_t DeviceSize`
- `__IO uint8_t MaxRdCurrentVDDMin`
- `__IO uint8_t MaxRdCurrentVDDMax`
- `__IO uint8_t MaxWrCurrentVDDMin`
- `__IO uint8_t MaxWrCurrentVDDMax`
- `__IO uint8_t DeviceSizeMul`
- `__IO uint8_t EraseGrSize`
- `__IO uint8_t EraseGrMul`
- `__IO uint8_t WrProtectGrSize`
- `__IO uint8_t WrProtectGrEnable`
- `__IO uint8_t ManDefIECC`
- `__IO uint8_t WrSpeedFact`
- `__IO uint8_t MaxWrBlockLen`
- `__IO uint8_t WriteBlockPaPartial`
- `__IO uint8_t Reserved3`
- `__IO uint8_t ContentProtectAppli`
- `__IO uint8_t FileFormatGrouop`
- `__IO uint8_t CopyFlag`
- `__IO uint8_t PermWrProtect`
- `__IO uint8_t TempWrProtect`
- `__IO uint8_t FileFormat`
- `__IO uint8_t ECC`
- `__IO uint8_t CSD_CRC`
- `__IO uint8_t Reserved4`

Field Documentation

- `__IO uint8_t HAL_SD_CardCSDTypedef::CSDStruct`
CSD structure
- `__IO uint8_t HAL_SD_CardCSDTypedef::SysSpecVersion`
System specification version
- `__IO uint8_t HAL_SD_CardCSDTypedef::Reserved1`
Reserved
- `__IO uint8_t HAL_SD_CardCSDTypedef::TAAC`
Data read access time 1
- `__IO uint8_t HAL_SD_CardCSDTypedef::NSAC`
Data read access time 2 in CLK cycles

- `__IO uint8_t HAL_SD_CardCSDTypedef::MaxBusClkFrec`
Max. bus clock frequency
- `__IO uint16_t HAL_SD_CardCSDTypedef::CardComdClasses`
Card command classes
- `__IO uint8_t HAL_SD_CardCSDTypedef::RdBlockLen`
Max. read data block length
- `__IO uint8_t HAL_SD_CardCSDTypedef::PartBlockRead`
Partial blocks for read allowed
- `__IO uint8_t HAL_SD_CardCSDTypedef::WrBlockMisalign`
Write block misalignment
- `__IO uint8_t HAL_SD_CardCSDTypedef::RdBlockMisalign`
Read block misalignment
- `__IO uint8_t HAL_SD_CardCSDTypedef::DSRImpl`
DSR implemented
- `__IO uint8_t HAL_SD_CardCSDTypedef::Reserved2`
Reserved
- `__IO uint32_t HAL_SD_CardCSDTypedef::DeviceSize`
Device Size
- `__IO uint8_t HAL_SD_CardCSDTypedef::MaxRdCurrentVDDMin`
Max. read current @ VDD min
- `__IO uint8_t HAL_SD_CardCSDTypedef::MaxRdCurrentVDDMax`
Max. read current @ VDD max
- `__IO uint8_t HAL_SD_CardCSDTypedef::MaxWrCurrentVDDMin`
Max. write current @ VDD min
- `__IO uint8_t HAL_SD_CardCSDTypedef::MaxWrCurrentVDDMax`
Max. write current @ VDD max
- `__IO uint8_t HAL_SD_CardCSDTypedef::DeviceSizeMul`
Device size multiplier
- `__IO uint8_t HAL_SD_CardCSDTypedef::EraseGrSize`
Erase group size
- `__IO uint8_t HAL_SD_CardCSDTypedef::EraseGrMul`
Erase group size multiplier
- `__IO uint8_t HAL_SD_CardCSDTypedef::WrProtectGrSize`
Write protect group size
- `__IO uint8_t HAL_SD_CardCSDTypedef::WrProtectGrEnable`
Write protect group enable
- `__IO uint8_t HAL_SD_CardCSDTypedef::ManDeflECC`
Manufacturer default ECC
- `__IO uint8_t HAL_SD_CardCSDTypedef::WrSpeedFact`
Write speed factor
- `__IO uint8_t HAL_SD_CardCSDTypedef::MaxWrBlockLen`

- Max. write data block length
- `__IO uint8_t HAL_SD_CardCSDTypedef::WriteBlockPaPartial`
Partial blocks for write allowed
- `__IO uint8_t HAL_SD_CardCSDTypedef::Reserved3`
Reserved
- `__IO uint8_t HAL_SD_CardCSDTypedef::ContentProtectAppli`
Content protection application
- `__IO uint8_t HAL_SD_CardCSDTypedef::FileFormatGrouop`
File format group
- `__IO uint8_t HAL_SD_CardCSDTypedef::CopyFlag`
Copy flag (OTP)
- `__IO uint8_t HAL_SD_CardCSDTypedef::PermWrProtect`
Permanent write protection
- `__IO uint8_t HAL_SD_CardCSDTypedef::TempWrProtect`
Temporary write protection
- `__IO uint8_t HAL_SD_CardCSDTypedef::FileFormat`
File format
- `__IO uint8_t HAL_SD_CardCSDTypedef::ECC`
ECC code
- `__IO uint8_t HAL_SD_CardCSDTypedef::CSD_CRC`
CSD CRC
- `__IO uint8_t HAL_SD_CardCSDTypedef::Reserved4`
Always 1

63.1.4 HAL_SD_CardCIDTypedef

Data Fields

- `__IO uint8_t ManufacturerID`
- `__IO uint16_t OEM_AppliID`
- `__IO uint32_t ProdName1`
- `__IO uint8_t ProdName2`
- `__IO uint8_t ProdRev`
- `__IO uint32_t ProdSN`
- `__IO uint8_t Reserved1`
- `__IO uint16_t ManufactDate`
- `__IO uint8_t CID_CRC`
- `__IO uint8_t Reserved2`

Field Documentation

- `__IO uint8_t HAL_SD_CardCIDTypedef::ManufacturerID`
Manufacturer ID
- `__IO uint16_t HAL_SD_CardCIDTypedef::OEM_AppliID`
OEM/Application ID
- `__IO uint32_t HAL_SD_CardCIDTypedef::ProdName1`

- Product Name part1
 - `__IO uint8_t HAL_SD_CardCIDTypeDef::ProdName2`
- Product Name part2
 - `__IO uint8_t HAL_SD_CardCIDTypeDef::ProdRev`
- Product Revision
 - `__IO uint32_t HAL_SD_CardCIDTypeDef::ProdSN`
- Product Serial Number
 - `__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved1`
- Reserved1
 - `__IO uint16_t HAL_SD_CardCIDTypeDef::ManufactDate`
- Manufacturing Date
 - `__IO uint8_t HAL_SD_CardCIDTypeDef::CID_CRC`
- CID CRC
 - `__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved2`
- Always 1

63.1.5 HAL_SD_CardStatusTypedef

Data Fields

- `__IO uint8_t DataBusWidth`
- `__IO uint8_t SecuredMode`
- `__IO uint16_t CardType`
- `__IO uint32_t ProtectedAreaSize`
- `__IO uint8_t SpeedClass`
- `__IO uint8_t PerformanceMove`
- `__IO uint8_t AllocationUnitSize`
- `__IO uint16_t EraseSize`
- `__IO uint8_t EraseTimeout`
- `__IO uint8_t EraseOffset`

Field Documentation

- `__IO uint8_t HAL_SD_CardStatusTypedef::DataBusWidth`
Shows the currently defined data bus width
- `__IO uint8_t HAL_SD_CardStatusTypedef::SecuredMode`
Card is in secured mode of operation
- `__IO uint16_t HAL_SD_CardStatusTypedef::CardType`
Carries information about card type
- `__IO uint32_t HAL_SD_CardStatusTypedef::ProtectedAreaSize`
Carries information about the capacity of protected area
- `__IO uint8_t HAL_SD_CardStatusTypedef::SpeedClass`
Carries information about the speed class of the card
- `__IO uint8_t HAL_SD_CardStatusTypedef::PerformanceMove`
Carries information about the card's performance move
- `__IO uint8_t HAL_SD_CardStatusTypedef::AllocationUnitSize`

- Carries information about the card's allocation unit size
- `__IO uint16_t HAL_SD_CardStatusTypedef::EraseSize`
Determines the number of AUs to be erased in one operation
- `__IO uint8_t HAL_SD_CardStatusTypedef::EraseTimeout`
Determines the timeout for any number of AU erase
- `__IO uint8_t HAL_SD_CardStatusTypedef::EraseOffset`
Carries information about the erase offset

63.2 SD Firmware driver API description

63.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in `HAL_SD_MspInit()` function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implement the `HAL_SD_MspInit()` API:
 - a. Enable the SDMMC interface clock using `__HAL_RCC_SDMMC_CLK_ENABLE()`;
 - b. SDMMC pins configuration for SD card
 - Enable the clock for the SDMMC GPIOs using the functions `__HAL_RCC_GPIOx_CLK_ENABLE()`;
 - Configure these SDMMC pins as alternate function pull-up using `HAL_GPIO_Init()` and according to your pin assignment;
 - c. NVIC configuration if you need to use interrupt process when using DMA transfer.
 - Configure the SDMMC interrupt priorities using functions `HAL_NVIC_SetPriority()`;
 - Enable the NVIC SDMMC IRQs using function `HAL_NVIC_EnableIRQ()`
 - SDMMC interrupts are managed using the macros `__HAL_SD_ENABLE_IT()` and `__HAL_SD_DISABLE_IT()` inside the communication process.
 - SDMMC interrupts pending bits are managed using the macros `__HAL_SD_GET_IT()` and `__HAL_SD_CLEAR_IT()`
 - d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC IP are used.
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

SD Card Initialization and configuration

To initialize the SD Card, use the `HAL_SD_Init()` function. It initializes the SD Card and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDMMC_CK) is computed as follows: $SDMMC_CK = SDMMCCCLK / (2 * ClockDiv)$ In initialization mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz.
2. Get the SD CID and CSD data. All these information are managed by the SDCardInfo structure. This structure provide also ready computed SD Card capacity and Block size.

Note:

These information are stored in SD handle structure in case of future use.

3. Configure the SD Card Data transfer frequency. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding SD Card according to the address read with the step 2.

5. Configure the SD Card in wide bus mode: 4-bits data.

SD Card Read operation

- You can read from SD card in polling mode by using function HAL_SD_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.
- You can read from SD card in DMA mode by using function HAL_SD_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.

SD Card Write operation

- You can write to SD card in polling mode by using function HAL_SD_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.
- You can write to SD card in DMA mode by using function HAL_SD_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.

SD card status

- At any time, you can check the SD Card status and get the SD card state by using the HAL_SD_GetStatusInfo() function. This function checks first if the SD card is still connected and then get the internal SD Card transfer state.

SD HAL driver macros list

Note:

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

63.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

This section contains the following APIs:

- [**HAL_SD_Init**](#)
- [**HAL_SD_InitCard**](#)
- [**HAL_SD_DelInit**](#)
- [**HAL_SD_MspInit**](#)
- [**HAL_SD_MspDelInit**](#)

63.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

This section contains the following APIs:

- [**HAL_SD_ReadBlocks**](#)
- [**HAL_SD_WriteBlocks**](#)
- [**HAL_SD_ReadBlocks_IT**](#)
- [**HAL_SD_WriteBlocks_IT**](#)
- [**HAL_SD_ReadBlocks_DMA**](#)
- [**HAL_SD_WriteBlocks_DMA**](#)
- [**HAL_SD_Erase**](#)
- [**HAL_SD_IRQHandler**](#)
- [**HAL_SD_GetState**](#)
- [**HAL_SD_GetError**](#)
- [**HAL_SD_TxCpltCallback**](#)

- [*HAL_SD_RxCpltCallback*](#)
- [*HAL_SD_ErrorCallback*](#)
- [*HAL_SD_AbortCallback*](#)
- [*HAL_SD_DriveTransceiver_1_8V_Callback*](#)

63.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations and get the related information

This section contains the following APIs:

- [*HAL_SD_GetCardCID*](#)
- [*HAL_SD_GetCardCSD*](#)
- [*HAL_SD_GetCardStatus*](#)
- [*HAL_SD_GetCardInfo*](#)
- [*HAL_SD_ConfigWideBusOperation*](#)
- [*HAL_SD_GetCardState*](#)
- [*HAL_SD_Abort*](#)
- [*HAL_SD_Abort_IT*](#)

63.2.5 Detailed description of functions

HAL_SD_Init

Function name

HAL_StatusTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd)

Function description

Initializes the SD according to the specified parameters in the SD_HandleTypeDef and create the associated handle.

Parameters

- **hsd:** Pointer to the SD handle

Return values

- **HAL:** status

HAL_SD_InitCard

Function name

HAL_StatusTypeDef HAL_SD_InitCard (SD_HandleTypeDef * hsd)

Function description

Initializes the SD Card according to the specified parameters in the.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **HAL:** status

Notes

- This function initializes the SD card. It could be used when a card re-initialization is needed.

HAL_SD_DelInit

Function name

`HAL_StatusTypeDef HAL_SD_DelInit (SD_HandleTypeDef * hsd)`

Function description

De-Initializes the SD card.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **HAL:** status

HAL_SD_MspInit

Function name

`void HAL_SD_MspInit (SD_HandleTypeDef * hsd)`

Function description

Initializes the SD MSP.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_MspDelInit

Function name

`void HAL_SD_MspDelInit (SD_HandleTypeDef * hsd)`

Function description

De-Initialize SD MSP.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_ReadBlocks

Function name

`HAL_StatusTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)`

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of SD blocks to read

- **Timeout:**

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_WriteBlocks

Function name

```
HAL_StatusTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd,  
uint32_t NumberOfBlocks, uint32_t Timeout)
```

Function description

Allows to write block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of SD blocks to write
- **Timeout:**

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_Erase

Function name

```
HAL_StatusTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint32_t BlockStartAdd, uint32_t  
BlockEndAdd)
```

Function description

Erases the specified memory area of the given SD card.

Parameters

- **hsd:** Pointer to SD handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_ReadBlocks_IT

Function name

```
HAL_StatusTypeDef HAL_SD_ReadBlocks_IT (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t  
BlockAdd, uint32_t NumberOfBlocks)
```

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_IT

Function name

```
HAL_StatusTypeDef HAL_SD_WriteBlocks_IT (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t
BlockAdd, uint32_t NumberOfBlocks)
```

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Tx interrupt event.

HAL_SD_ReadBlocks_DMA

Function name

```
HAL_StatusTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t
BlockAdd, uint32_t NumberOfBlocks)
```

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_DMA

Function name

```
HAL_StatusTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)
```

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Tx interrupt event.

HAL_SD_IRQHandler

Function name

```
void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)
```

Function description

This function handles SD card interrupt request.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_TxCpltCallback

Function name

```
void HAL_SD_TxCpltCallback (SD_HandleTypeDef * hsd)
```

Function description

Tx Transfer completed callbacks.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_RxCpltCallback

Function name

```
void HAL_SD_RxCpltCallback (SD_HandleTypeDef * hsd)
```

Function description

Rx Transfer completed callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_ErrorCallback

Function name

```
void HAL_SD_ErrorCallback (SD_HandleTypeDef * hsd)
```

Function description

SD error callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_AbortCallback

Function name

```
void HAL_SD_AbortCallback (SD_HandleTypeDef * hsd)
```

Function description

SD Abort callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_DriveTransciver_1_8V_Callback

Function name

```
void HAL_SD_DriveTransciver_1_8V_Callback (FlagStatus status)
```

Function description

Enable/Disable the SD Transciver 1.8V Mode Callback.

Parameters

- **status:** Voltage Switch State

Return values

- **None:**

HAL_SD_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigWideBusOperation (SD_HandleTypeDef * hsd, uint32_t WideMode)

Function description

Enables wide bus operation for the requested card if supported by card.

Parameters

- **hsd:** Pointer to SD handle
- **WideMode:** Specifies the SD card wide bus mode This parameter can be one of the following values:
 - SDMMC_BUS_WIDE_8B: 8-bit data transfer
 - SDMMC_BUS_WIDE_4B: 4-bit data transfer
 - SDMMC_BUS_WIDE_1B: 1-bit data transfer

Return values

- **HAL:** status

HAL_SD_SendSDStatus

Function name

HAL_StatusTypeDef HAL_SD_SendSDStatus (SD_HandleTypeDef * hsd, uint32_t * pSDstatus)

Function description

HAL_SD_GetCardState

Function name

HAL_SD_CardStateTypedef HAL_SD_GetCardState (SD_HandleTypeDef * hsd)

Function description

Gets the current sd card data state.

Parameters

- **hsd:** pointer to SD handle

Return values

- **Card:** state

HAL_SD_GetCardCID

Function name

HAL_StatusTypeDef HAL_SD_GetCardCID (SD_HandleTypeDef * hsd, HAL_SD_CardCIDTypedef * pCID)

Function description

Returns information the information of the card which are stored on the CID register.

Parameters

- **hsd:** Pointer to SD handle
- **pCID:** Pointer to a HAL_SD_CIDTypedef structure that contains all CID register parameters

Return values

- **HAL:** status

HAL_SD_GetCardCSD

Function name

```
HAL_StatusTypeDef HAL_SD_GetCardCSD (SD_HandleTypeDef * hsd, HAL_SD_CardCSDTypeDef * pCSD)
```

Function description

Returns information the information of the card which are stored on the CSD register.

Parameters

- **hsd:** Pointer to SD handle
- **pCSD:** Pointer to a HAL_SD_CardInfoTypedef structure that contains all CSD register parameters

Return values

- **HAL:** status

HAL_SD_GetCardStatus

Function name

```
HAL_StatusTypeDef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pStatus)
```

Function description

Gets the SD status info.

Parameters

- **hsd:** Pointer to SD handle
- **pStatus:** Pointer to the HAL_SD_CardStatusTypedef structure that will contain the SD card status information

Return values

- **HAL:** status

HAL_SD_GetCardInfo

Function name

```
HAL_StatusTypeDef HAL_SD_GetCardInfo (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * pCardInfo)
```

Function description

Gets the SD card info.

Parameters

- **hsd:** Pointer to SD handle
- **pCardInfo:** Pointer to the HAL_SD_CardInfoTypeDef structure that will contain the SD card status information

Return values

- **HAL:** status

HAL_SD_GetState

Function name

```
HAL_SD_StateTypeDef HAL_SD_GetState (SD_HandleTypeDef * hsd)
```

Function description

return the SD state

Parameters

- **hsd:** Pointer to sd handle

Return values

- **HAL:** state

HAL_SD_GetError**Function name**

`uint32_t HAL_SD_GetError (SD_HandleTypeDef * hsd)`

Function description

Return the SD error code.

Parameters

- **hsd:** : pointer to a SD_HandleTypeDef structure that contains the configuration information.

Return values

- **SD:** Error Code

HAL_SD_Abort**Function name**

`HAL_StatusTypeDef HAL_SD_Abort (SD_HandleTypeDef * hsd)`

Function description

Abort the current transfer and disable the SD.

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.

Return values

- **HAL:** status

HAL_SD_Abort_IT**Function name**

`HAL_StatusTypeDef HAL_SD_Abort_IT (SD_HandleTypeDef * hsd)`

Function description

Abort the current transfer and disable the SD (IT mode).

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.

Return values

- **HAL:** status

63.3 SD Firmware driver defines

63.3.1 SD

SD Error status enumeration Structure definition

HAL_SD_ERROR_NONE

No error

HAL_SD_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_SD_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_SD_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_SD_ERROR_DATA_TIMEOUT

Data timeout

HAL_SD_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_SD_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_SD_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_SD_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_SD_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_SD_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_SD_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_SD_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_SD_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_SD_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_SD_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_SD_ERROR_CC_ERR

Internal card controller error

HAL_SD_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_SD_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_SD_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_SD_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_SD_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_SD_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_SD_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_SD_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_SD_ERROR_INVALID_VOLRANGE

Error in case of invalid voltage range

HAL_SD_ERROR_ADDR_OUT_OF_RANGE

Error when addressed block is out of range

HAL_SD_ERROR_REQUEST_NOT_APPLICABLE

Error when command request is not applicable

HAL_SD_ERROR_PARAM

the used parameter is not valid

HAL_SD_ERROR_UNSUPPORTED_FEATURE

Error when feature is not insupported

HAL_SD_ERROR_BUSY

Error when transfer process is busy

HAL_SD_ERROR_DMA

Error while DMA transfer

HAL_SD_ERROR_TIMEOUT

Timeout Error

SD context enumeration**SD_CONTEXT_NONE**

None

SD_CONTEXT_READ_SINGLE_BLOCK

Read single block operation

SD_CONTEXT_READ_MULTIPLE_BLOCK

Read multiple blocks operation

SD_CONTEXT_WRITE_SINGLE_BLOCK

Write single block operation

SD_CONTEXT_WRITE_MULTIPLE_BLOCK

Write multiple blocks operation

SD_CONTEXT_IT

Process in Interrupt mode

SD_CONTEXT_DMA

Process in DMA mode

SD Supported Memory Cards**CARD_NORMAL_SPEED**

Normal Speed Card <12.5Mo/s , Spec Version 1.01

CARD_HIGH_SPEED

High Speed Card <25Mo/s , Spec version 2.00

CARD_ULTRA_HIGH_SPEED

UHS-I SD Card <50Mo/s for SDR50, DDR5 Cards and <104Mo/s for SDR104, Spec version 3.01

CARD_SDSC

SD Standard Capacity <2Go

CARD_SDHC_SDXC

SD High Capacity <32Go, SD Extended Capacity <2To

CARD_SECURED***SD Supported Version*****CARD_V1_X****CARD_V2_X*****Exported Constants*****BLOCKSIZE**

Block size is 512 bytes

SD Exported Macros**_HAL_SD_ENABLE_IT****Description:**

- Enable the SD device interrupt.

Parameters:

- **_HANDLE_**: SD Handle
- **_INTERRUPT_**: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
 - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
 - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
 - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
 - SDMMC_IT_DTIMEOUT: Data timeout interrupt
 - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt

- SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
- SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
- SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
- SDMMC_IT_DHOLD: Data transfer Hold interrupt
- SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
- SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
- SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
- SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
- SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
- SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
- SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
- SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
- SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
- SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

HAL_SD_DISABLE_IT

Description:

- Disable the SD device interrupt.

Parameters:

- HANDLE: SD Handle
- INTERRUPT: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
 - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
 - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
 - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
 - SDMMC_IT_DTIMEOUT: Data timeout interrupt
 - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
 - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
 - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
 - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
 - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
 - SDMMC_IT_DHOLD: Data transfer Hold interrupt
 - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
 - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
 - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
 - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
 - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
 - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
 - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
 - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
 - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
 - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
 - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt

- SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
- SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

[__HAL_SD_GET_FLAG](#)**Description:**

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

Parameters:

- __HANDLE__: SD Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
 - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
 - SDMMC_FLAG_CTIMEOUT: Command response timeout
 - SDMMC_FLAG_DTIMEOUT: Data timeout
 - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
 - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
 - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
 - SDMMC_FLAG_CMDSENT: Command sent (no response required)
 - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
 - SDMMC_FLAG_DHOLD: Data transfer Hold
 - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
 - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
 - SDMMC_FLAG_CPSMACT: Command path state machine active
 - SDMMC_FLAG_DPSMACT: Data path state machine active
 - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
 - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
 - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
 - SDMMC_FLAG_RXFIFOF: Receive FIFO full
 - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
 - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
 - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
 - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
 - SDMMC_FLAG_SDIOIT: SDIO interrupt received
 - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
 - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
 - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
 - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
 - SDMMC_FLAG_IDMATE: IDMA transfer error
 - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of SD FLAG (SET or RESET).

[__HAL_SD_CLEAR_FLAG](#)**Description:**

- Clear the SD's pending flags.

Parameters:

- __HANDLE__: SD Handle

- `__FLAG__`: specifies the flag to clear. This parameter can be one or a combination of the following values:
 - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
 - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
 - SDMMC_FLAG_CTIMEOUT: Command response timeout
 - SDMMC_FLAG_DTIMEOUT: Data timeout
 - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
 - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
 - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
 - SDMMC_FLAG_CMDSENT: Command sent (no response required)
 - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
 - SDMMC_FLAG_DHOLD: Data transfer Hold
 - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
 - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
 - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
 - SDMMC_FLAG_SDIOIT: SDIO interrupt received
 - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
 - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
 - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
 - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
 - SDMMC_FLAG_IDMATE: IDMA transfer error
 - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

[_HAL_SD_GET_IT](#)**Description:**

- Check whether the specified SD interrupt has occurred or not.

Parameters:

- `__HANDLE__`: SD Handle
- `__INTERRUPT__`: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
 - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
 - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
 - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
 - SDMMC_IT_DTIMEOUT: Data timeout interrupt
 - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
 - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
 - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
 - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
 - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
 - SDMMC_IT_DHOLD: Data transfer Hold interrupt
 - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
 - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
 - SDMMC_IT_DPSMACT: Data path state machine active interrupt
 - SDMMC_IT_CPSMACT: Command path state machine active interrupt
 - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
 - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
 - SDMMC_IT_TXFIFOF: Transmit FIFO full interrupt
 - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt

- SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDMMC_IT_RXFIFOE: Receive FIFO empty interrupt
- SDMMC_IT_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
- SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
- SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
- SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
- SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
- SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
- SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
- SDMMC_IT_IDMATE: IDMA transfer error interrupt
- SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of SD IT (SET or RESET).

[_HAL_SD_CLEAR_IT](#)

Description:

- Clear the SD's interrupt pending bits.

Parameters:

- __HANDLE__: SD Handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
 - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
 - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
 - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
 - SDMMC_IT_DTIMEOUT: Data timeout interrupt
 - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
 - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
 - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
 - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
 - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
 - SDMMC_IT_DHOLD: Data transfer Hold interrupt
 - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
 - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
 - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
 - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
 - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
 - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
 - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
 - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
 - SDMMC_IT_IDMATE: IDMA transfer error interrupt
 - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

SD Handle Structure definition**[SD_InitTypeDef](#)****[SD_TypeDef](#)**

64 HAL SD Extension Driver

64.1 SDEx Firmware driver API description

64.1.1 How to use this driver

The SD Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using `HAL_SDEEx_ConfigDMAMultiBuffer()` function.
- Start Read and Write for multibuffer mode using `HAL_SDEEx_ReadBlocksDMAMultiBuffer()` and `HAL_SDEEx_WriteBlocksDMAMultiBuffer()` functions.

64.1.2 Multibuffer functions

This section provides functions allowing to configure the multibuffer mode and start read and write multibuffer mode for SD HAL driver.

This section contains the following APIs:

- [`HAL_SDEEx_ConfigDMAMultiBuffer`](#)
- [`HAL_SDEEx_ReadBlocksDMAMultiBuffer`](#)
- [`HAL_SDEEx_WriteBlocksDMAMultiBuffer`](#)
- [`HAL_SDEEx_ChangeDMABuffer`](#)
- [`HAL_SDEEx_Read_DMADoubleBuffer0CpltCallback`](#)
- [`HAL_SDEEx_Read_DMADoubleBuffer1CpltCallback`](#)
- [`HAL_SDEEx_Write_DMADoubleBuffer0CpltCallback`](#)
- [`HAL_SDEEx_Write_DMADoubleBuffer1CpltCallback`](#)

64.1.3 Detailed description of functions

`HAL_SDEEx_ConfigDMAMultiBuffer`

Function name

```
HAL_StatusTypeDef HAL_SDEEx_ConfigDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t *  
pDataBuffer0, uint32_t * pDataBuffer1, uint32_t BufferSize)
```

Function description

Configure DMA Dual Buffer mode.

Parameters

- **hsd:** SD handle
- **pDataBuffer0:** Pointer to the buffer0 that will contain/receive the transferred data
- **pDataBuffer1:** Pointer to the buffer1 that will contain/receive the transferred data
- **BufferSize:** Size of Buffer0 in Blocks. Buffer0 and Buffer1 must have the same size.

Return values

- **HAL:** status

`HAL_SDEEx_ReadBlocksDMAMultiBuffer`

Function name

```
HAL_StatusTypeDef HAL_SDEEx_ReadBlocksDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t  
BlockAdd, uint32_t NumberOfBlocks)
```

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** SD handle
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Total number of blocks to read

Return values

- **HAL:** status

HAL_SDEEx_WriteBlocksDMAMultiBuffer

Function name

```
HAL_StatusTypeDef HAL_SDEEx_WriteBlocksDMAMultiBuffer (SD_HandleTypeDef * hsd, uint32_t
BlockAdd, uint32_t NumberOfBlocks)
```

Function description

Write block(s) to a specified address in a card.

Parameters

- **hsd:** SD handle
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Total number of blocks to read

Return values

- **HAL:** status

HAL_SDEEx_ChangeDMABuffer

Function name

```
HAL_StatusTypeDef HAL_SDEEx_ChangeDMABuffer (SD_HandleTypeDef * hsd,
HAL_SDEEx_DMABuffer_MemoryTypeDef Buffer, uint32_t * pDataBuffer)
```

Function description

Change the DMA Buffer0 or Buffer1 address on the fly.

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure.
- **Buffer:** the buffer to be changed, This parameter can be one of the following values: SD_DMA_BUFFER0 or SD_DMA_BUFFER1
- **pDataBuffer:** The new address

Return values

- **HAL:** status

Notes

- The BUFFER0 address can be changed only when the current transfer use BUFFER1 and the BUFFER1 address can be changed only when the current transfer use BUFFER0.

HAL_SDEEx_Read_DMADoubleBuffer0CpltCallback

Function name

```
void HAL_SDEEx_Read_DMADoubleBuffer0CpltCallback (SD_HandleTypeDef * hsd)
```

Function description

Read DMA Buffer 0 Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

HAL_SDEEx_Read_DMADoubleBuffer1CpltCallback

Function name

void HAL_SDEEx_Read_DMADoubleBuffer1CpltCallback (SD_HandleTypeDef * hsd)

Function description

Read DMA Buffer 1 Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

HAL_SDEEx_Write_DMADoubleBuffer0CpltCallback

Function name

void HAL_SDEEx_Write_DMADoubleBuffer0CpltCallback (SD_HandleTypeDef * hsd)

Function description

Write DMA Buffer 0 Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

HAL_SDEEx_Write_DMADoubleBuffer1CpltCallback

Function name

void HAL_SDEEx_Write_DMADoubleBuffer1CpltCallback (SD_HandleTypeDef * hsd)

Function description

Write DMA Buffer 1 Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

65 HAL SMARTCARD Generic Driver

65.1 SMARTCARD Firmware driver registers structures

65.1.1 SMARTCARD_InitTypeDef

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t StopBits*
- *uint16_t Parity*
- *uint16_t Mode*
- *uint16_t CLKPolarity*
- *uint16_t CLKPhase*
- *uint16_t CLKLastBit*
- *uint16_t OneBitSampling*
- *uint8_t Prescaler*
- *uint8_t GuardTime*
- *uint16_t NACKEnable*
- *uint32_t TimeOutEnable*
- *uint32_t TimeOutValue*
- *uint8_t BlockLength*
- *uint8_t AutoRetryCount*
- *uint32_t FIFOMode*
- *uint32_t TXFIFOThreshold*
- *uint32_t RXFIFOThreshold*

Field Documentation

- *uint32_t SMARTCARD_InitTypeDef::BaudRate*

Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / (hsmartcard->Init.BaudRate))

- *uint32_t SMARTCARD_InitTypeDef::WordLength*

Specifies the number of data bits transmitted or received in a frame. This parameter **SMARTCARD Word Length** can only be set to 9 (8 data + 1 parity bits).

- *uint32_t SMARTCARD_InitTypeDef::StopBits*

Specifies the number of stop bits. This parameter can be a value of **SMARTCARD Number of Stop Bits**.

- *uint16_t SMARTCARD_InitTypeDef::Parity*

Specifies the parity mode. This parameter can be a value of **SMARTCARD Parity**

Note:

- The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
 - *uint16_t SMARTCARD_InitTypeDef::Mode*
- Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of **SMARTCARD Transfer Mode**
- *uint16_t SMARTCARD_InitTypeDef::CLKPolarity*

Specifies the steady state of the serial clock. This parameter can be a value of **SMARTCARD Clock Polarity**

- **uint16_t SMARTCARD_InitTypeDef::CLKPhase**

Specifies the clock transition on which the bit capture is made. This parameter can be a value of **SMARTCARD Clock Phase**

- **uint16_t SMARTCARD_InitTypeDef::CLKLastBit**

Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of **SMARTCARD Last Bit**

- **uint16_t SMARTCARD_InitTypeDef::OneBitSampling**

Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of **SMARTCARD One Bit Sampling Method**.

- **uint8_t SMARTCARD_InitTypeDef::Prescaler**

Specifies the SmartCard Prescaler.

- **uint8_t SMARTCARD_InitTypeDef::GuardTime**

Specifies the SmartCard Guard Time applied after stop bits.

- **uint16_t SMARTCARD_InitTypeDef::NACKEnable**

Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of **SMARTCARD NACK Enable**

- **uint32_t SMARTCARD_InitTypeDef::TimeOutEnable**

Specifies whether the receiver timeout is enabled. This parameter can be a value of **SMARTCARD Timeout Enable**

- **uint32_t SMARTCARD_InitTypeDef::TimeOutValue**

Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.

- **uint8_t SMARTCARD_InitTypeDef::BlockLength**

Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF

- **uint8_t SMARTCARD_InitTypeDef::AutoRetryCount**

Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)

- **uint32_t SMARTCARD_InitTypeDef::FIFOMode**

Specifies if the FIFO mode will be used. This parameter can be a value of **SMARTCARD FIFO mode**

- **uint32_t SMARTCARD_InitTypeDef::TXFIFOThreshold**

Specifies the TXFIFO threshold level. This parameter can be a value of **SMARTCARD TXFIFO threshold level**

- **uint32_t SMARTCARD_InitTypeDef::RXFIFOThreshold**

Specifies the RXFIFO threshold level. This parameter can be a value of **SMARTCARD RXFIFO threshold level**

65.1.2 SMARTCARD_AdvFeatureInitTypeDef

Data Fields

- **uint32_t AdvFeatureInit**
- **uint32_t TxPinLevelInvert**
- **uint32_t RxPinLevelInvert**
- **uint32_t DataInvert**

- `uint32_t Swap`
- `uint32_t OverrunDisable`
- `uint32_t DMADisableonRxError`
- `uint32_t MSBFirst`
- `uint16_t TxCompletionIndication`

Field Documentation

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit`

Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of **SMARTCARD advanced feature initialization type**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert`

Specifies whether the TX pin active level is inverted. This parameter can be a value of **SMARTCARD advanced feature TX pin active level inversion**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert`

Specifies whether the RX pin active level is inverted. This parameter can be a value of **SMARTCARD advanced feature RX pin active level inversion**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert`

Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of **SMARTCARD advanced feature Binary Data inversion**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap`

Specifies whether TX and RX pins are swapped. This parameter can be a value of **SMARTCARD advanced feature RX TX pins swap**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable`

Specifies whether the reception overrun detection is disabled. This parameter can be a value of **SMARTCARD advanced feature Overrun Disable**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError`

Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of **SMARTCARD advanced feature DMA Disable on Rx Error**

- `uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst`

Specifies whether MSB is sent first on UART line. This parameter can be a value of **SMARTCARD advanced feature MSB first**

- `uint16_t SMARTCARD_AdvFeatureInitTypeDef::TxCompletionIndication`

Specifies which transmission completion indication is used: before (when relevant flag is available) or once guard time period has elapsed. This parameter can be a value of **SMARTCARD Transmission Completion Indication**.

65.1.3 SMARTCARD_HandleTypeDef

Data Fields

- `USART_TypeDef * Instance`
- `SMARTCARD_InitTypeDef Init`
- `SMARTCARD_AdvFeatureInitTypeDef AdvancedInit`
- `uint8_t * pTxBuffPtr`
- `uint16_t TxXferSize`
- `__IO uint16_t TxXferCount`
- `uint8_t * pRxBuffPtr`
- `uint16_t RxXferSize`
- `__IO uint16_t RxXferCount`

- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `_IO HAL_SMARTCARD_StateTypeDef gState`
- `_IO HAL_SMARTCARD_StateTypeDef RxState`
- `uint32_t ErrorCode`

Field Documentation

- **`USART_TypeDef* SMARTCARD_HandleTypeDef::Instance`**
USART registers base address
- **`SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init`**
SmartCard communication parameters
- **`SMARTCARD_AdvFeatureInitTypeDef SMARTCARD_HandleTypeDef::AdvancedInit`**
SmartCard advanced features initialization parameters
- **`uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr`**
Pointer to SmartCard Tx transfer Buffer
- **`uint16_t SMARTCARD_HandleTypeDef::TxXferSize`**
SmartCard Tx Transfer size
- **`_IO uint16_t SMARTCARD_HandleTypeDef::TxXferCount`**
SmartCard Tx Transfer Counter
- **`uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr`**
Pointer to SmartCard Rx transfer Buffer
- **`uint16_t SMARTCARD_HandleTypeDef::RxXferSize`**
SmartCard Rx Transfer size
- **`_IO uint16_t SMARTCARD_HandleTypeDef::RxXferCount`**
SmartCard Rx Transfer Counter
- **`DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx`**
SmartCard Tx DMA Handle parameters
- **`DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx`**
SmartCard Rx DMA Handle parameters
- **`HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock`**
Locking object
- **`_IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::gState`**
SmartCard state information related to global Handle management and also related to Tx operations. This parameter can be a value of `HAL_SMARTCARD_StateTypeDef`
- **`_IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::RxState`**
SmartCard state information related to Rx operations. This parameter can be a value of `HAL_SMARTCARD_StateTypeDef`
- **`uint32_t SMARTCARD_HandleTypeDef::ErrorCode`**
SmartCard Error code

65.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmcard).
2. Associate a USART to the SMARTCARD handle hsmcard.
3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
 - Enable the USARTx interface clock.
 - USART pins configuration:
 - Enable the clock for the USART GPIOs.
 - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
 - NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
 - Configure the USARTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
 - DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmcard handle AdvancedInit structure.
6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.

Note:

The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

SMARTCARD HAL driver macros list

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

- __HAL_SMARTCARD_GET_FLAG : Check whether or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled

Note: You can refer to the SMARTCARD HAL driver header file for more useful macros

65.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
 - Baud Rate
 - Parity: parity should be enabled, frame Length is fixed to 8 bits plus parity
 - Receiver/transmitter modes
 - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
 - Prescaler value
 - Guard bit time
 - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
 - TX and/or RX pin level inversion
 - data logical level inversion
 - RX and TX pins swap
 - RX overrun detection disabling
 - DMA disabling on RX error
 - MSB first on communication line
 - Time out enabling (and if activated, timeout value)
 - Block length
 - Auto-retry counter

The HAL_SMARTCARD_Init() API follows the USART synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [**HAL_SMARTCARD_Init**](#)
- [**HAL_SMARTCARD_DelInit**](#)
- [**HAL_SMARTCARD_MspInit**](#)
- [**HAL_SMARTCARD_MspDelInit**](#)

65.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMARTCARD data transfers.

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART_CR2 register.

(+) There are two modes of transfer: (++) Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer. (++) Non-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. (++) The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected. (+) Blocking mode APIs are : (++) HAL_SMARTCARD_Transmit() (++) HAL_SMARTCARD_Receive() (+) Non Blocking mode APIs with Interrupt are : (++) HAL_SMARTCARD_Transmit_IT() (++) HAL_SMARTCARD_Receive_IT() (++) HAL_SMARTCARD_IRQHandler() (+) Non Blocking mode functions with DMA are : (++) HAL_SMARTCARD_Transmit_DMA() (++) HAL_SMARTCARD_Receive_DMA() (+) A set of Transfer Complete Callbacks are provided in non Blocking mode: (++) HAL_SMARTCARD_TxCpltCallback() (++) HAL_SMARTCARD_RxCpltCallback() (++) HAL_SMARTCARD_ErrorCallback()

1. Non-Blocking mode transfers could be aborted using Abort API's : (+) HAL_SMARTCARD_Abort() (+) HAL_SMARTCARD_AbortTransmit() (+) HAL_SMARTCARD_AbortReceive() (+) HAL_SMARTCARD_Abort_IT() (+) HAL_SMARTCARD_AbortTransmit_IT() (+) HAL_SMARTCARD_AbortReceive_IT()
 2. For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided: (+) HAL_SMARTCARD_AbortCpltCallback() (+) HAL_SMARTCARD_AbortTransmitCpltCallback() (+) HAL_SMARTCARD_AbortReceiveCpltCallback()
 3. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
(+) Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed. Transfer is kept ongoing on SMARTCARD side. If user wants to abort it, Abort services should be called by user. (+) Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Frame Error in Interrupt mode transmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.
- There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - Non-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
 - The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.
 - Blocking mode APIs are :
 - HAL_SMARTCARD_Transmit()
 - HAL_SMARTCARD_Receive()
 - Non Blocking mode APIs with Interrupt are :
 - HAL_SMARTCARD_Transmit_IT()
 - HAL_SMARTCARD_Receive_IT()
 - HAL_SMARTCARD_IRQHandler()

- Non Blocking mode functions with DMA are :
 - HAL_SMARTCARD_Transmit_DMA()
 - HAL_SMARTCARD_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_SMARTCARD_TxCpltCallback()
 - HAL_SMARTCARD_RxCpltCallback()
 - HAL_SMARTCARD_ErrorCallback() (#) Non-Blocking mode transfers could be aborted using Abort API's :
- HAL_SMARTCARD_Abort()
- HAL_SMARTCARD_AbortTransmit()
- HAL_SMARTCARD_AbortReceive()
- HAL_SMARTCARD_Abort_IT()
- HAL_SMARTCARD_AbortTransmit_IT()
- HAL_SMARTCARD_AbortReceive_IT() (#) For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
- HAL_SMARTCARD_AbortCpltCallback()
- HAL_SMARTCARD_AbortTransmitCpltCallback()
- HAL_SMARTCARD_AbortReceiveCpltCallback() (#) In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
- Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed. Transfer is kept ongoing on SMARTCARD side. If user wants to abort it, Abort services should be called by user.
- Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Frame Error in Interrupt mode transmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

This section contains the following APIs:

- [HAL_SMARTCARD_Transmit](#)
- [HAL_SMARTCARD_Receive](#)
- [HAL_SMARTCARD_Transmit_IT](#)
- [HAL_SMARTCARD_Receive_IT](#)
- [HAL_SMARTCARD_Transmit_DMA](#)
- [HAL_SMARTCARD_Receive_DMA](#)
- [HAL_SMARTCARD_Abort](#)
- [HAL_SMARTCARD_AbortTransmit](#)
- [HAL_SMARTCARD_AbortReceive](#)
- [HAL_SMARTCARD_Abort_IT](#)
- [HAL_SMARTCARD_AbortTransmit_IT](#)
- [HAL_SMARTCARD_AbortReceive_IT](#)
- [HAL_SMARTCARD_IRQHandler](#)
- [HAL_SMARTCARD_TxCpltCallback](#)
- [HAL_SMARTCARD_RxCpltCallback](#)
- [HAL_SMARTCARD_ErrorCallback](#)
- [HAL_SMARTCARD_AbortCpltCallback](#)
- [HAL_SMARTCARD_AbortTransmitCpltCallback](#)
- [HAL_SMARTCARD_AbortReceiveCpltCallback](#)

65.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard handle and also return Peripheral Errors occurred during communication process

- HAL_SMARTCARD_GetState() API can be helpful to check in run-time the state of the SMARTCARD peripheral.
- HAL_SMARTCARD_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:

- [HAL_SMARTCARD_GetState](#)
- [HAL_SMARTCARD_GetError](#)

65.2.5 Detailed description of functions

HAL_SMARTCARD_Init

Function name

`HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)`

Function description

Initialize the SMARTCARD mode according to the specified parameters in the SMARTCARD_HandleTypeDef and initialize the associated handle.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_DelInit

Function name

`HAL_StatusTypeDef HAL_SMARTCARD_DelInit (SMARTCARD_HandleTypeDef * hsmartcard)`

Function description

Deinitialize the SMARTCARD peripheral.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_MspInit

Function name

`void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)`

Function description

Initialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_MspDeInit

Function name

void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Deinitialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_Transmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_SMARTCARD_Receive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_SMARTCARD_Transmit_IT

Function name

```
HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard,  
uint8_t * pData, uint16_t Size)
```

Function description

Send an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_SMARTCARD_Receive_IT

Function name

```
HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t  
* pData, uint16_t Size)
```

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

HAL_SMARTCARD_Transmit_DMA

Function name

```
HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard,  
uint8_t * pData, uint16_t Size)
```

Function description

Send an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_SMARTCARD_Receive_DMA

Function name

```
HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard,  
uint8_t * pData, uint16_t Size)
```

Function description

Receive an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

HAL_SMARTCARD_Abort

Function name

```
HAL_StatusTypeDef HAL_SMARTCARD_Abort (SMARTCARD_HandleTypeDef * hsmartcard)
```

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortTransmit

Function name

```
HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit (SMARTCARD_HandleTypeDef * hsmartcard)
```

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortReceive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortTransmit_IT

Function name

`HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsmartcard)`

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortReceive_IT

Function name

`HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsmartcard)`

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_IRQHandler

Function name

`void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)`

Function description

Handle SMARTCARD interrupt requests.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_TxCpltCallback

Function name

void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Tx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_RxCpltCallback

Function name

void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Rx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_ErrorCallback

Function name

void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD error callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortCpltCallback

Function name

void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name

void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name

void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Receive Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_GetState

Function name

HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle state.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle state

HAL_SMARTCARD_GetError

Function name

uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle error code.

Parameters

- **hsmcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle Error Code

65.3 SMARTCARD Firmware driver defines

65.3.1 SMARTCARD

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD frame phase on first clock transition

SMARTCARD_PHASE_2EDGE

SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD frame low polarity

SMARTCARD_POLARITY_HIGH

SMARTCARD frame high polarity

SMARTCARD auto retry counter LSB position in CR3 register

SMARTCARD_CR3_SCARCNT_LSB_POS

SMARTCARD auto retry counter LSB position in CR3 register

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

SMARTCARD_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

SMARTCARD Exported Macros

_HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle state.

Parameters:

- `__HANDLE__`: SMARTCARD handle.

Return value:

- None

[__HAL_SMARTCARD_FLUSH_DRREGISTER](#)**Description:**

- Flush the Smartcard Data registers.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

[__HAL_SMARTCARD_CLEAR_FLAG](#)**Description:**

- Clear the specified SMARTCARD pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be any combination of the following values:
 - SMARTCARD_CLEAR_PEF Parity error clear flag
 - SMARTCARD_CLEAR_FEF Framing error clear flag
 - SMARTCARD_CLEAR_NEF Noise detected clear flag
 - SMARTCARD_CLEAR_OREF OverRun error clear flag
 - SMARTCARD_CLEAR_IDLEF Idle line detected clear flag
 - SMARTCARD_CLEAR_TCF Transmission complete clear flag
 - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag
 - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
 - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

[__HAL_SMARTCARD_CLEAR_PEFLAG](#)**Description:**

- Clear the SMARTCARD PE pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

[__HAL_SMARTCARD_CLEAR_FEFLAG](#)**Description:**

- Clear the SMARTCARD FE pending flag.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_OREFLAG

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_IDLEFLAG

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - SMARTCARD_FLAG_TCBGT Transmission complete before guard time flag
 - SMARTCARD_FLAG_REACK Receive enable acknowledge flag
 - SMARTCARD_FLAG_TEACK Transmit enable acknowledge flag
 - SMARTCARD_FLAG_BUSY Busy flag
 - SMARTCARD_FLAG_EOBF End of block flag
 - SMARTCARD_FLAG_RTOF Receiver timeout flag
 - SMARTCARD_FLAG_TXE Transmit data register empty flag
 - SMARTCARD_FLAG_TC Transmission complete flag
 - SMARTCARD_FLAG_RXNE Receive data register not empty flag
 - SMARTCARD_FLAG_IDLE Idle line detection flag
 - SMARTCARD_FLAG_ORE Overrun error flag
 - SMARTCARD_FLAG_NE Noise error flag
 - SMARTCARD_FLAG_FE Framing error flag
 - SMARTCARD_FLAG_PE Parity error flag

Return value:

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

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
 - SMARTCARD_IT_EOB End of block interrupt
 - SMARTCARD_IT_RTO Receive timeout interrupt
 - SMARTCARD_IT_TXE Transmit data register empty interrupt
 - SMARTCARD_IT_TC Transmission complete interrupt
 - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
 - SMARTCARD_IT_RXNE Receive data register not empty interrupt
 - SMARTCARD_IT_IDLE Idle line detection interrupt
 - SMARTCARD_IT_PE Parity error interrupt
 - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)

Return value:

- None

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
 - SMARTCARD_IT_EOB End of block interrupt
 - SMARTCARD_IT_RTO Receive timeout interrupt
 - SMARTCARD_IT_TXE Transmit data register empty interrupt
 - SMARTCARD_IT_TC Transmission complete interrupt
 - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
 - SMARTCARD_IT_RXNE Receive data register not empty interrupt
 - SMARTCARD_IT_IDLE Idle line detection interrupt
 - SMARTCARD_IT_PE Parity error interrupt
 - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)

Return value:

- None

__HAL_SMARTCARD_GET_IT

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT__: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
 - SMARTCARD_IT_EOB End of block interrupt
 - SMARTCARD_IT_RTO Receive timeout interrupt

- SMARTCARD_IT_TXE Transmit data register empty interrupt
- SMARTCARD_IT_TC Transmission complete interrupt
- SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
- SMARTCARD_IT_RXNE Receive data register not empty interrupt
- SMARTCARD_IT_IDLE Idle line detection interrupt
- SMARTCARD_IT_ORE Overrun error interrupt
- SMARTCARD_IT_NE Noise error interrupt
- SMARTCARD_IT_FE Framing error interrupt
- SMARTCARD_IT_PE Parity error interrupt

Return value:

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

[_HAL_SMARTCARD_GET_IT_SOURCE](#)

Description:

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

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
 - SMARTCARD_IT_EOB End of block interrupt
 - SMARTCARD_IT_RTO Receive timeout interrupt
 - SMARTCARD_IT_TXE Transmit data register empty interrupt
 - SMARTCARD_IT_TC Transmission complete interrupt
 - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
 - SMARTCARD_IT_RXNE Receive data register not empty interrupt
 - SMARTCARD_IT_IDLE Idle line detection interrupt
 - SMARTCARD_IT_ERR Framing, overrun or noise error interrupt
 - SMARTCARD_IT_PE Parity error interrupt

Return value:

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

[_HAL_SMARTCARD_CLEAR_IT](#)

Description:

- Clear the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
 - SMARTCARD_CLEAR_PEF Parity error clear flag
 - SMARTCARD_CLEAR_FEF Framing error clear flag
 - SMARTCARD_CLEAR_NEF Noise detected clear flag
 - SMARTCARD_CLEAR_OREF OverRun error clear flag
 - SMARTCARD_CLEAR_IDLEF Idle line detection clear flag
 - SMARTCARD_CLEAR_TCF Transmission complete clear flag
 - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag (when flag available)
 - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag

- SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

[__HAL_SMARTCARD_CLEAR_TXFECF](#)**Description:**

- Clear the SMARTCARD TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

[__HAL_SMARTCARD_SEND_REQ](#)**Description:**

- Set a specific SMARTCARD request flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
 - SMARTCARD_RXDATA_FLUSH_REQUEST Receive data flush Request
 - SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

[__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE](#)**Description:**

- Enable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

[__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE](#)**Description:**

- Disable the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

[__HAL_SMARTCARD_ENABLE](#)**Description:**

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

_HAL_SMARTCARD_DISABLE**Description:**

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

SMARTCARD FIFO mode**SMARTCARD_FIFOMODE_DISABLE**

FIFO mode disable

SMARTCARD_FIFOMODE_ENABLE

FIFO mode enable

SMARTCARD guard time value LSB position in GTPR register**SMARTCARD_GTPR_GT_LSB_POS**

SMARTCARD guard time value LSB position in GTPR register

SMARTCARD interruptions flags mask**SMARTCARD_IT_MASK**

SMARTCARD interruptions flags mask

SMARTCARD Last Bit**SMARTCARD_LASTBIT_DISABLE**

SMARTCARD frame last data bit clock pulse not output to SCLK pin

SMARTCARD_LASTBIT_ENABLE

SMARTCARD frame last data bit clock pulse output to SCLK pin

SMARTCARD Transfer Mode**SMARTCARD_MODE_RX**

SMARTCARD RX mode

SMARTCARD_MODE_TX

SMARTCARD TX mode

SMARTCARD_MODE_TX_RX

SMARTCARD RX and TX mode

SMARTCARD advanced feature MSB first**SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE**

Most significant bit sent/received first disable

SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

SMARTCARD NACK Enable

SMARTCARD_NACK_ENABLE

SMARTCARD NACK transmission disabled

SMARTCARD_NACK_DISABLE

SMARTCARD NACK transmission enabled

SMARTCARD One Bit Sampling Method**SMARTCARD_ONE_BIT_SAMPLE_DISABLE**

SMARTCARD frame one-bit sample disabled

SMARTCARD_ONE_BIT_SAMPLE_ENABLE

SMARTCARD frame one-bit sample enabled

SMARTCARD advanced feature Overrun Disable**SMARTCARD_ADVFEATURE_OVERRUN_ENABLE**

RX overrun enable

SMARTCARD_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

SMARTCARD Parity**SMARTCARD_PARITY_EVEN**

SMARTCARD frame even parity

SMARTCARD_PARITY_ODD

SMARTCARD frame odd parity

SMARTCARD Request Parameters**SMARTCARD_RXDATA_FLUSH_REQUEST**

Receive data flush request

SMARTCARD_TXDATA_FLUSH_REQUEST

Transmit data flush request

SMARTCARD block length LSB position in RTOR register**SMARTCARD_RTOR_BLEN_LSB_POS**

SMARTCARD block length LSB position in RTOR register

SMARTCARD RXFIFO threshold level**SMARTCARD_RXFIFO_THRESHOLD_1_8**

RXFIFO threshold 1 eighth full configuration

SMARTCARD_RXFIFO_THRESHOLD_1_4

RXFIFO threshold 1 quart full configuration

SMARTCARD_RXFIFO_THRESHOLD_1_2

RXFIFO threshold half full configuration

SMARTCARD_RXFIFO_THRESHOLD_3_4

RXFIFO threshold 3 quarts full configuration

SMARTCARD_RXFIFO_THRESHOLD_7_8

RXFIFO threshold 7 eighth full configuration

SMARTCARD_RXFIFO_THRESHOLD_8_8

RXFIFO becomes Full

SMARTCARD advanced feature RX pin active level inversion**SMARTCARD_ADVFEATURE_RXINV_DISABLE**

RX pin active level inversion disable

SMARTCARD_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap**SMARTCARD_ADVFEATURE_SWAP_DISABLE**

TX/RX pins swap disable

SMARTCARD_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

SMARTCARD Number of Stop Bits**SMARTCARD_STOPBITS_0_5**

SMARTCARD frame with 0.5 stop bit

SMARTCARD_STOPBITS_1_5

SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable**SMARTCARD_TIMEOUT_DISABLE**

SMARTCARD receiver timeout disabled

SMARTCARD_TIMEOUT_ENABLE

SMARTCARD receiver timeout enabled

SMARTCARD TXFIFO threshold level**SMARTCARD_TXFIFO_THRESHOLD_1_8**

TXFIFO threshold 1 eighth full configuration

SMARTCARD_TXFIFO_THRESHOLD_1_4

TXFIFO threshold 1 quart full configuration

SMARTCARD_TXFIFO_THRESHOLD_1_2

TXFIFO threshold half full configuration

SMARTCARD_TXFIFO_THRESHOLD_3_4

TXFIFO threshold 3 quarts full configuration

SMARTCARD_TXFIFO_THRESHOLD_7_8

TXFIFO threshold 7 eighth full configuration

SMARTCARD_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

SMARTCARD advanced feature TX pin active level inversion**SMARTCARD_ADVFEATURE_TXINV_DISABLE**

TX pin active level inversion disable

SMARTCARD_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

SMARTCARD Word Length**SMARTCARD_WORDLENGTH_9B**

SMARTCARD frame length

66 HAL SMARTCARD Extension Driver

66.1 SMARTCARDEEx Firmware driver API description

66.1.1 SMARTCARD peripheral extended features

The Extended SMARTCARD HAL driver can be used as follows:

1. After having configured the SMARTCARD basic features with HAL_SMARTCARD_Init(), then program SMARTCARD advanced features if required (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.

66.1.2 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARDEEx_BlockLength_Config() API allows to configure the Block Length on the fly
- HAL_SMARTCARDEEx_TimeOut_Config() API allows to configure the receiver timeout value on the fly
- HAL_SMARTCARDEEx_EnableReceiverTimeOut() API enables the receiver timeout feature
- HAL_SMARTCARDEEx_DisableReceiverTimeOut() API disables the receiver timeout feature

This section contains the following APIs:

- [HAL_SMARTCARDEEx_BlockLength_Config](#)
- [HAL_SMARTCARDEEx_TimeOut_Config](#)
- [HAL_SMARTCARDEEx_EnableReceiverTimeOut](#)
- [HAL_SMARTCARDEEx_DisableReceiverTimeOut](#)

66.1.3 Detailed description of functions

HAL_SMARTCARDEEx_BlockLength_Config

Function name

```
void HAL_SMARTCARDEEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)
```

Function description

Update on the fly the SMARTCARD block length in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **BlockLength:** SMARTCARD block length (8-bit long at most)

Return values

- **None:**

HAL_SMARTCARDEEx_TimeOut_Config

Function name

```
void HAL_SMARTCARDEEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)
```

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **TimeOutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0xFFFFFFFF.

Return values

- **None:**

HAL_SMARTCARDEX_EnableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEX_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEX_DisableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEX_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

66.2 SMARTCARDEX Firmware driver defines

66.2.1 SMARTCARDEX

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT

No advanced feature initialization

SMARTCARD_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

SMARTCARD_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

SMARTCARD_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

SMARTCARD_ADVFEATURE_SWAP_INIT

TX/RX pins swap

SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

SMARTCARD_ADVFEATURE_TXCOMPLETION

TX completion indication before or after guard time

SMARTCARD Flags**SMARTCARD_FLAG_TCBGT**

SMARTCARD transmission complete before guard time completion

SMARTCARD_FLAG_REACK

SMARTCARD receive enable acknowledge flag

SMARTCARD_FLAG_TEACK

SMARTCARD transmit enable acknowledge flag

SMARTCARD_FLAG_BUSY

SMARTCARD busy flag

SMARTCARD_FLAG_EOBF

SMARTCARD end of block flag

SMARTCARD_FLAG_RTOF

SMARTCARD receiver timeout flag

SMARTCARD_FLAG_TXE

SMARTCARD transmit data register empty

SMARTCARD_FLAG_TC

SMARTCARD transmission complete

SMARTCARD_FLAG_RXNE

SMARTCARD read data register not empty

SMARTCARD_FLAG_IDLE

SMARTCARD idle line detection

SMARTCARD_FLAG_ORE

SMARTCARD overrun error

SMARTCARD_FLAG_NE

SMARTCARD noise error

SMARTCARD_FLAG_FE

SMARTCARD frame error

SMARTCARD_FLAG_PE

SMARTCARD parity error

SMARTCARD_FLAG_TXFT

SMARTCARD TXFIFO threshold flag

SMARTCARD_FLAG_RXFT

SMARTCARD RXFIFO threshold flag

SMARTCARD_FLAG_RXFF

SMARTCARD RXFIFO Fullflag

SMARTCARD_FLAG_TXFE

SMARTCARD TXFIFO Empty flag

SMARTCARD Interrupts Definition**SMARTCARD_IT_PE**

SMARTCARD parity error interruption

SMARTCARD_IT_TXE

SMARTCARD transmit data register empty interruption

SMARTCARD_IT_TC

SMARTCARD transmission complete interruption

SMARTCARD_IT_RXNE

SMARTCARD read data register not empty interruption

SMARTCARD_IT_IDLE

SMARTCARD idle line detection interruption

SMARTCARD_IT_ERR

SMARTCARD error interruption

SMARTCARD_IT_ORE

SMARTCARD overrun error interruption

SMARTCARD_IT_NE

SMARTCARD noise error interruption

SMARTCARD_IT_FE

SMARTCARD frame error interruption

SMARTCARD_IT_EOB

SMARTCARD end of block interruption

SMARTCARD_IT_RTO

SMARTCARD receiver timeout interruption

SMARTCARD_IT_RXFF**SMARTCARD_IT_TXFE**

SMARTCARD_IT_RXFT

SMARTCARD_IT_TXFT

SMARTCARD_IT_TCBGT

SMARTCARD transmission complete before guard time completion interruption

SMARTCARD Interruption Clear Flags

SMARTCARD_CLEAR_PEF

SMARTCARD parity error clear flag

SMARTCARD_CLEAR_FEF

SMARTCARD framing error clear flag

SMARTCARD_CLEAR_NEF

SMARTCARD noise detected clear flag

SMARTCARD_CLEAR_OREF

SMARTCARD overrun error clear flag

SMARTCARD_CLEAR_IDLEF

SMARTCARD idle line detected clear flag

SMARTCARD_CLEAR_TCF

SMARTCARD transmission complete clear flag

SMARTCARD_CLEAR_TCBGTF

SMARTCARD transmission complete before guard time completion clear flag

SMARTCARD_CLEAR_RTOF

SMARTCARD receiver time out clear flag

SMARTCARD_CLEAR_EOBF

SMARTCARD end of block clear flag

SMARTCARD_CLEAR_TXFECF

SMARTCARD TXFIFO empty clear flag

SMARTCARD_CLEAR_UDRCF

SMARTCARD UnderRun Error Clear Flag

SMARTCARD Transmission Completion Indication

SMARTCARD_TCBGT

SMARTCARD transmission complete before guard time

SMARTCARD_TC

SMARTCARD transmission complete (flag raised when guard time has elapsed)

67 HAL SMBUS Generic Driver

67.1 SMBUS Firmware driver registers structures

67.1.1 SMBUS_InitTypeDef

Data Fields

- *uint32_t Timing*
- *uint32_t AnalogFilter*
- *uint32_t OwnAddress1*
- *uint32_t AddressingMode*
- *uint32_t DualAddressMode*
- *uint32_t OwnAddress2*
- *uint32_t OwnAddress2Masks*
- *uint32_t GeneralCallMode*
- *uint32_t NoStretchMode*
- *uint32_t PacketErrorCheckMode*
- *uint32_t PeripheralMode*
- *uint32_t SMBusTimeout*

Field Documentation

- *uint32_t SMBUS_InitTypeDef::Timing*

Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual

- *uint32_t SMBUS_InitTypeDef::AnalogFilter*

Specifies if Analog Filter is enable or not. This parameter can be a value of **SMBUS Analog Filter**

- *uint32_t SMBUS_InitTypeDef::OwnAddress1*

Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.

- *uint32_t SMBUS_InitTypeDef::AddressingMode*

Specifies if 7-bit or 10-bit addressing mode for master is selected. This parameter can be a value of **SMBUS addressing mode**

- *uint32_t SMBUS_InitTypeDef::DualAddressMode*

Specifies if dual addressing mode is selected. This parameter can be a value of **SMBUS dual addressing mode**

- *uint32_t SMBUS_InitTypeDef::OwnAddress2*

Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.

- *uint32_t SMBUS_InitTypeDef::OwnAddress2Masks*

Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of **SMBUS own address2 masks**.

- *uint32_t SMBUS_InitTypeDef::GeneralCallMode*

Specifies if general call mode is selected. This parameter can be a value of **SMBUS general call addressing mode**.

- *uint32_t SMBUS_InitTypeDef::NoStretchMode*

Specifies if nostretch mode is selected. This parameter can be a value of **SMBUS nostretch mode**

- **uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode**

Specifies if Packet Error Check mode is selected. This parameter can be a value of **SMBUS packet error check mode**

- **uint32_t SMBUS_InitTypeDef::PeripheralMode**

Specifies which mode of Periphal is selected. This parameter can be a value of **SMBUS peripheral mode**

- **uint32_t SMBUS_InitTypeDef::SMBusTimeout**

Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

67.1.2 SMBUS_HandleTypeDef

Data Fields

- **I2C_TypeDef * Instance**
- **SMBUS_InitTypeDef Init**
- **uint8_t * pBuffPtr**
- **uint16_t XferSize**
- **__IO uint16_t XferCount**
- **__IO uint32_t XferOptions**
- **__IO uint32_t PreviousState**
- **HAL_LockTypeDef Lock**
- **__IO uint32_t State**
- **__IO uint32_t ErrorCode**

Field Documentation

- **I2C_TypeDef* SMBUS_HandleTypeDef::Instance**
SMBUS registers base address
- **SMBUS_InitTypeDef SMBUS_HandleTypeDef::Init**
SMBUS communication parameters
- **uint8_t* SMBUS_HandleTypeDef::pBuffPtr**
Pointer to SMBUS transfer buffer
- **uint16_t SMBUS_HandleTypeDef::XferSize**
SMBUS transfer size
- **__IO uint16_t SMBUS_HandleTypeDef::XferCount**
SMBUS transfer counter
- **__IO uint32_t SMBUS_HandleTypeDef::XferOptions**
SMBUS transfer options
- **__IO uint32_t SMBUS_HandleTypeDef::PreviousState**
SMBUS communication Previous state
- **HAL_LockTypeDef SMBUS_HandleTypeDef::Lock**
SMBUS locking object
- **__IO uint32_t SMBUS_HandleTypeDef::State**
SMBUS communication state
- **__IO uint32_t SMBUS_HandleTypeDef::ErrorCode**
SMBUS Error code

67.2 SMBUS Firmware driver API description

67.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example: SMBUS_HandleTypeDef hsmbus;
2. Initialize the SMBUS low level resources by implementing the HAL_SMBUS_MspInit() API:
 - Enable the SMBUSx interface clock
 - SMBUS pins configuration
 - Enable the clock for the SMBUS GPIOs
 - Configure SMBUS pins as alternate function open-drain
 - NVIC configuration if you need to use interrupt process
 - Configure the SMBUSx interrupt priority
 - Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Addressing Mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call, Nostretch mode, Peripheral mode and Packet Error Check mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the HAL_SMBUS_Init() API:
 - These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Transmit_IT()
 - At transmission end of transfer HAL_SMBUS_MasterTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_MasterTxCpltCallback()
- Receive in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Receive_IT()
 - At reception end of transfer HAL_SMBUS_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_MasterRxCpltCallback()
- Abort a master/host SMBUS process communication with Interrupt using HAL_SMBUS_Master_Abort_IT()
 - The associated previous transfer callback is called at the end of abort process
 - mean HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit
 - mean HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive
- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using HAL_SMBUS_EnableListen_IT() HAL_SMBUS_DisableListen_IT()
 - When address slave/device SMBUS match, HAL_SMBUS_AddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).
 - At Listen mode end HAL_SMBUS_ListenCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_ListenCpltCallback()
- Transmit in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Transmit_IT()
 - At transmission end of transfer HAL_SMBUS_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveTxCpltCallback()
- Receive in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Receive_IT()
 - At reception end of transfer HAL_SMBUS_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback()

- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT()
HAL_SMBUS_DisableAlert_IT()
 - When SMBUS Alert is generated HAL_SMBUS_ErrorCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function HAL_SMBUS_GetError()
- Get HAL state machine or error values using HAL_SMBUS_GetState() or HAL_SMBUS_GetError()
- In case of transfer Error, HAL_SMBUS_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Error Code using function HAL_SMBUS_GetError()

SMBUS HAL driver macros list

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

- __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- __HAL_SMBUS_GET_FLAG : Checks whether the specified SMBUS flag is set or not
- __HAL_SMBUS_CLEAR_FLAG : Clears the specified SMBUS pending flag
- __HAL_SMBUS_ENABLE_IT: Enables the specified SMBUS interrupt
- __HAL_SMBUS_DISABLE_IT: Disables the specified SMBUS interrupt

Note: You can refer to the SMBUS HAL driver header file for more useful macros

67.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SMBUSx peripheral:

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
 - Clock Timing
 - Bus Timeout
 - Analog Filter mode
 - Own Address 1
 - Addressing mode (Master, Slave)
 - Dual Addressing mode
 - Own Address 2
 - Own Address 2 Mask
 - General call mode
 - Nostretch mode
 - Packet Error Check mode
 - Peripheral mode
- Call the function HAL_SMBUS_DelInit() to restore the default configuration of the selected SMBUSx peripheral.

This section contains the following APIs:

- [**HAL_SMBUS_Init**](#)
- [**HAL_SMBUS_DelInit**](#)
- [**HAL_SMBUS_MspInit**](#)
- [**HAL_SMBUS_MspDelInit**](#)

67.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMBUS data transfers.

1. Blocking mode function to check if device is ready for usage is :

- HAL_SMBUS_IsDeviceReady()
2. There is only one mode of transfer:
- Non-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. Non-Blocking mode functions with Interrupt are :
- HAL_SMBUS_Master_Transmit_IT()
 - HAL_SMBUS_Master_Receive_IT()
 - HAL_SMBUS_Slave_Transmit_IT()
 - HAL_SMBUS_Slave_Receive_IT()
 - HAL_SMBUS_EnableListen_IT()
 - HAL_SMBUS_DisableListen_IT()
 - HAL_SMBUS_EnableAlert_IT()
 - HAL_SMBUS_DisableAlert_IT()
4. A set of Transfer Complete Callbacks are provided in Non_Blocking mode:
- HAL_SMBUS_MasterTxCpltCallback()
 - HAL_SMBUS_MasterRxCpltCallback()
 - HAL_SMBUS_SlaveTxCpltCallback()
 - HAL_SMBUS_SlaveRxCpltCallback()
 - HAL_SMBUS_AddrCallback()
 - HAL_SMBUS_ListenCpltCallback()
 - HAL_SMBUS_ErrorCallback()

This section contains the following APIs:

- [**HAL_SMBUS_Master_Transmit_IT**](#)
- [**HAL_SMBUS_Master_Receive_IT**](#)
- [**HAL_SMBUS_Master_Abort_IT**](#)
- [**HAL_SMBUS_Slave_Transmit_IT**](#)
- [**HAL_SMBUS_Slave_Receive_IT**](#)
- [**HAL_SMBUS_EnableListen_IT**](#)
- [**HAL_SMBUS_DisableListen_IT**](#)
- [**HAL_SMBUS_EnableAlert_IT**](#)
- [**HAL_SMBUS_DisableAlert_IT**](#)
- [**HAL_SMBUS_IsDeviceReady**](#)

67.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- [**HAL_SMBUS_GetState**](#)
- [**HAL_SMBUS_GetError**](#)

67.2.5 Detailed description of functions

HAL_SMBUS_Init

Function name

HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and initialize the associated handle.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DelInit

Function name

```
HAL_StatusTypeDef HAL_SMBUS_DelInit (SMBUS_HandleTypeDef * hsmbus)
```

Function description

DeInitialize the SMBUS peripheral.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_MspInit

Function name

```
void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)
```

Function description

Initialize the SMBUS MSP.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MspDelInit

Function name

```
void HAL_SMBUS_MspDelInit (SMBUS_HandleTypeDef * hsmbus)
```

Function description

DeInitialize the SMBUS MSP.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_IsDeviceReady

Function name

```
HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)
```

Function description

Check if target device is ready for communication.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used with Memory devices

HAL_SMBUS_Master_Transmit_IT

Function name

`HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)`

Function description

Transmit in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Receive_IT

Function name

`HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)`

Function description

Receive in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Abort_IT

Function name

```
HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)
```

Function description

Abort a master/host SMBUS process communication with Interrupt.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address

Return values

- **HAL:** status

Notes

- This abort can be called only if state is ready

HAL_SMBUS_Slave_Transmit_IT

Function name

```
HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)
```

Function description

Transmit in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Slave_Receive_IT

Function name

```
HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)
```

Function description

Receive in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_EnableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_DisableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_EV_IRQHandler

Function name

void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS event interrupt request.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ER_IRQHandler

Function name

void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS error interrupt request.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterTxCpltCallback

Function name

void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Tx Transfer completed callback.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterRxCpltCallback

Function name

void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Rx Transfer completed callback.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveTxCpltCallback

Function name

void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveRxCpltCallback

Function name

void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_AddrCallback

Function name

void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **TransferDirection:** Master request Transfer Direction (Write/Read)
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_SMBUS_ListenCpltCallback

Function name

```
void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)
```

Function description

Listen Complete callback.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ErrorCallback

Function name

```
void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)
```

Function description

SMBUS error callback.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_GetState

Function name

```
uint32_t HAL_SMBUS_GetState (SMBUS_HandleTypeDef * hsmbus)
```

Function description

Return the SMBUS handle state.

Parameters

- **hsmbus:** : Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** state

HAL_SMBUS_GetError

Function name

```
uint32_t HAL_SMBUS_GetError (SMBUS_HandleTypeDef * hsmbus)
```

Function description

Return the SMBUS error code.

Parameters

- **hsmbus:** : pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **SMBUS:** Error Code

67.3 SMBUS Firmware driver defines

67.3.1 SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS_ADDRESSINGMODE_10BIT

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

SMBUS Error Code definition

HAL_SMBUS_ERROR_NONE

No error

HAL_SMBUS_ERROR_BERR

BERR error

HAL_SMBUS_ERROR_ARLO

ARLO error

HAL_SMBUS_ERROR_ACKF

ACKF error

HAL_SMBUS_ERROR_OVR

OVR error

HAL_SMBUS_ERROR_HALTIMEOUT

Timeout error

HAL_SMBUS_ERROR_BUSTIMEOUT

Bus Timeout error

HAL_SMBUS_ERROR_ALERT

Alert error

HAL_SMBUS_ERROR_PECERR

PEC error

SMBUS Exported Macros

_HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.

Return value:

- None

`__HAL_SMBUS_ENABLE_IT`

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable. This parameter can be one of the following values:
 - `SMBUS_IT_ERRI`: Errors interrupt enable
 - `SMBUS_IT_TCI`: Transfer complete interrupt enable
 - `SMBUS_IT_STOPI`: STOP detection interrupt enable
 - `SMBUS_IT_NACKI`: NACK received interrupt enable
 - `SMBUS_IT_ADDRI`: Address match interrupt enable
 - `SMBUS_IT_RXI`: RX interrupt enable
 - `SMBUS_IT_TXI`: TX interrupt enable

Return value:

- None

`__HAL_SMBUS_DISABLE_IT`

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.
- `__INTERRUPT__`: specifies the interrupt source to disable. This parameter can be one of the following values:
 - `SMBUS_IT_ERRI`: Errors interrupt enable
 - `SMBUS_IT_TCI`: Transfer complete interrupt enable
 - `SMBUS_IT_STOPI`: STOP detection interrupt enable
 - `SMBUS_IT_NACKI`: NACK received interrupt enable
 - `SMBUS_IT_ADDRI`: Address match interrupt enable
 - `SMBUS_IT_RXI`: RX interrupt enable
 - `SMBUS_IT_TXI`: TX interrupt enable

Return value:

- None

`__HAL_SMBUS_GET_IT_SOURCE`

Description:

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

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.
- `__INTERRUPT__`: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:

- SMBUS_IT_ERRI: Errors interrupt enable
- SMBUS_IT_TCI: Transfer complete interrupt enable
- SMBUS_IT_STOPI: STOP detection interrupt enable
- SMBUS_IT_NACKI: NACK received interrupt enable
- SMBUS_IT_ADDRI: Address match interrupt enable
- SMBUS_IT_RXI: RX interrupt enable
- SMBUS_IT_TXI: TX interrupt enable

Return value:

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

SMBUS_FLAG_MASK

Description:

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

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - SMBUS_FLAG_TXE: Transmit data register empty
 - SMBUS_FLAG_TXIS: Transmit interrupt status
 - SMBUS_FLAG_RXNE: Receive data register not empty
 - SMBUS_FLAG_ADDR: Address matched (slave mode)
 - SMBUS_FLAG_AF: NACK received flag
 - SMBUS_FLAG_STOPF: STOP detection flag
 - SMBUS_FLAG_TC: Transfer complete (master mode)
 - SMBUS_FLAG_TCR: Transfer complete reload
 - SMBUS_FLAG_BERR: Bus error
 - SMBUS_FLAG_ARLO: Arbitration lost
 - SMBUS_FLAG_OVR: Overrun/Underrun
 - SMBUS_FLAG_PECERR: PEC error in reception
 - SMBUS_FLAG_TIMEOUT: Timeout or Tlow detection flag
 - SMBUS_FLAG_ALERT: SMBus alert
 - SMBUS_FLAG_BUSY: Bus busy
 - SMBUS_FLAG_DIR: Transfer direction (slave mode)

Return value:

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

__HAL_SMBUS_GET_FLAG

__HAL_SMBUS_CLEAR_FLAG

Description:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
 - SMBUS_FLAG_ADDR: Address matched (slave mode)
 - SMBUS_FLAG_AF: NACK received flag
 - SMBUS_FLAG_STOPF: STOP detection flag
 - SMBUS_FLAG_BERR: Bus error
 - SMBUS_FLAG_ARLO: Arbitration lost

- SMBUS_FLAG_OVR: Overrun/Underrun
- SMBUS_FLAG_PECERR: PEC error in reception
- SMBUS_FLAG_TIMEOUT: Timeout or Tlow detection flag
- SMBUS_FLAG_ALERT: SMBus alert

Return value:

- None

[__HAL_SMBUS_ENABLE](#)**Description:**

- Enable the specified SMBUS peripheral.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.

Return value:

- None

[__HAL_SMBUS_DISABLE](#)**Description:**

- Disable the specified SMBUS peripheral.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.

Return value:

- None

[__HAL_SMBUS_GENERATE_NACK](#)**Description:**

- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:

- `__HANDLE__`: specifies the SMBUS Handle.

Return value:

- None

SMBUS Flag definition[SMBUS_FLAG_TXE](#)[SMBUS_FLAG_RXIS](#)[SMBUS_FLAG_RXNE](#)[SMBUS_FLAG_ADDR](#)[SMBUS_FLAG_AF](#)[SMBUS_FLAG_STOPF](#)[SMBUS_FLAG_TC](#)[SMBUS_FLAG_TCR](#)

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLE

SMBUS_GENERALCALL_ENABLE

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLE

SMBUS_NOSTRETCH_ENABLE

SMBUS own address2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

SMBUS_OTHER_FRAME_NO_PEC

SMBUS_OTHER_FRAME_WITH_PEC

SMBUS_OTHER_AND_LAST_FRAME_NO_PEC

SMBUS_OTHER_AND_LAST_FRAME_WITH_PEC

68 HAL SPDIFRX Generic Driver

68.1 SPDIFRX Firmware driver registers structures

68.1.1 SPDIFRX_InitTypeDef

Data Fields

- *uint32_t InputSelection*
- *uint32_t Retries*
- *uint32_t WaitForActivity*
- *uint32_t ChannelSelection*
- *uint32_t DataFormat*
- *uint32_t StereoMode*
- *uint32_t PreambleTypeMask*
- *uint32_t ChannelStatusMask*
- *uint32_t ValidityBitMask*
- *uint32_t ParityErrorMask*
- *FunctionalState SymbolClockGen*
- *FunctionalState BackupSymbolClockGen*

Field Documentation

- *uint32_t SPDIFRX_InitTypeDef::InputSelection*

Specifies the SPDIF input selection. This parameter can be a value of **SPDIFRX Input Selection**

- *uint32_t SPDIFRX_InitTypeDef::Retries*

Specifies the Maximum allowed re-tries during synchronization phase. This parameter can be a value of **SPDIFRX Maximum Retries**

- *uint32_t SPDIFRX_InitTypeDef::WaitForActivity*

Specifies the wait for activity on SPDIF selected input. This parameter can be a value of **SPDIFRX Wait For Activity**.

- *uint32_t SPDIFRX_InitTypeDef::ChannelSelection*

Specifies whether the control flow will take the channel status from channel A or B. This parameter can be a value of **SPDIFRX Channel Selection**

- *uint32_t SPDIFRX_InitTypeDef::DataFormat*

Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of **SPDIFRX Data Format**

- *uint32_t SPDIFRX_InitTypeDef::StereoMode*

Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of **SPDIFRX Stereo Mode**

- *uint32_t SPDIFRX_InitTypeDef::PreambleTypeMask*

Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of **SPDIFRX Preamble Type Mask**

- *uint32_t SPDIFRX_InitTypeDef::ChannelStatusMask*

Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of **SPDIFRX Channel Status Mask**

- *uint32_t SPDIFRX_InitTypeDef::ValidityBitMask*

Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of **SPDIFRX Validity Mask**

- `uint32_t SPDIFRX_InitTypeDef::ParityErrorMask`

Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of **SPDIFRX Parity Error Mask**

- `FunctionalState SPDIFRX_InitTypeDef::SymbolClockGen`

Enable/Disable the SPDIFRX Symbol Clock generation. This parameter can be set to Enable or Disable

- `FunctionalState SPDIFRX_InitTypeDef::BackupSymbolClockGen`

Enable/Disable the SPDIFRX Backup Symbol Clock generation. This parameter can be set to Enable or Disable

68.1.2 SPDIFRX_SetDataFormatTypeDef

Data Fields

- `uint32_t DataFormat`
- `uint32_t StereoMode`
- `uint32_t PreambleTypeMask`
- `uint32_t ChannelStatusMask`
- `uint32_t ValidityBitMask`
- `uint32_t ParityErrorMask`

Field Documentation

- `uint32_t SPDIFRX_SetDataFormatTypeDef::DataFormat`

Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of **SPDIFRX Data Format**

- `uint32_t SPDIFRX_SetDataFormatTypeDef::StereoMode`

Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of **SPDIFRX Stereo Mode**

- `uint32_t SPDIFRX_SetDataFormatTypeDef::PreambleTypeMask`

Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of **SPDIFRX Preamble Type Mask**

- `uint32_t SPDIFRX_SetDataFormatTypeDef::ChannelStatusMask`

Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of **SPDIFRX Channel Status Mask**

- `uint32_t SPDIFRX_SetDataFormatTypeDef::ValidityBitMask`

Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of **SPDIFRX Validity Mask**

- `uint32_t SPDIFRX_SetDataFormatTypeDef::ParityErrorMask`

Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of **SPDIFRX Parity Error Mask**

68.1.3 SPDIFRX_HandleTypeDef

Data Fields

- `SPDIFRX_TypeDef * Instance`
- `SPDIFRX_InitTypeDef Init`
- `uint32_t * pRxBuffPtr`
- `uint32_t * pCsBuffPtr`
- `__IO uint16_t RxXferSize`

- `__IO uint16_t RxXferCount`
- `__IO uint16_t CsXferSize`
- `__IO uint16_t CsXferCount`
- `DMA_HandleTypeDef * hdmaCsRx`
- `DMA_HandleTypeDef * hdmaDrRx`
- `__IO HAL_LockTypeDef Lock`
- `__IO HAL_SPDIFRX_StateTypeDef State`
- `__IO uint32_t ErrorCode`

Field Documentation

- `SPDIFRX_TypeDef* SPDIFRX_HandleTypeDef::Instance`
- `SPDIFRX_InitTypeDef SPDIFRX_HandleTypeDef::Init`
- `uint32_t* SPDIFRX_HandleTypeDef::pRxBuffPtr`
- `uint32_t* SPDIFRX_HandleTypeDef::pCsBuffPtr`
- `__IO uint16_t SPDIFRX_HandleTypeDef::RxXferSize`
- `__IO uint16_t SPDIFRX_HandleTypeDef::RxXferCount`
- `__IO uint16_t SPDIFRX_HandleTypeDef::CsXferSize`
- `__IO uint16_t SPDIFRX_HandleTypeDef::CsXferCount`
- `DMA_HandleTypeDef* SPDIFRX_HandleTypeDef::hdmaCsRx`
- `DMA_HandleTypeDef* SPDIFRX_HandleTypeDef::hdmaDrRx`
- `__IO HAL_LockTypeDef SPDIFRX_HandleTypeDef::Lock`
- `__IO HAL_SPDIFRX_StateTypeDef SPDIFRX_HandleTypeDef::State`
- `__IO uint32_t SPDIFRX_HandleTypeDef::ErrorCode`

68.2 SPDIFRX Firmware driver API description

68.2.1 How to use this driver

The SPDIFRX HAL driver can be used as follow:

1. Declare SPDIFRX_HandleTypeDef handle structure.
2. Initialize the SPDIFRX low level resources by implement the HAL_SPDIFRX_Msplinit() API:
 - a. Enable the SPDIFRX interface clock.
 - b. SPDIFRX pins configuration:
 - Enable the clock for the SPDIFRX GPIOs.
 - Configure these SPDIFRX pins as alternate function pull-up.
 - c. NVIC configuration if you need to use interrupt process (HAL_SPDIFRX_ReceiveControlFlow_IT() and HAL_SPDIFRX_ReceiveDataFlow_IT() API's).
 - Configure the SPDIFRX interrupt priority.
 - Enable the NVIC SPDIFRX IRQ handle.
 - d. DMA Configuration if you need to use DMA process (HAL_SPDIFRX_ReceiveDataFlow_DMA() and HAL_SPDIFRX_ReceiveControlFlow_DMA() API's).

- Declare a DMA handle structure for the reception of the Data Flow channel.
 - Declare a DMA handle structure for the reception of the Control Flow channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure CtrlRx/DataRx with the required parameters.
 - Configure the DMA Channel.
 - Associate the initialized DMA handle to the SPDIFRX DMA CtrlRx/DataRx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA CtrlRx/ DataRx channel.
3. Program the input selection, re-tries number, wait for activity, channel status selection, data format, stereo mode and masking of user bits using HAL_SPDIFRX_Init() function.

Note: *The specific SPDIFRX interrupts (RXNE/CSRNE and Error Interrupts) will be managed using the macros __SPDIFRX_ENABLE_IT() and __SPDIFRX_DISABLE_IT() inside the receive process.*

Note: *Make sure that ck_spdif clock is configured.*

4. Three operation modes are available within this driver :

Polling mode for reception operation (for debug purpose)

- Receive data flow in blocking mode using HAL_SPDIFRX_ReceiveDataFlow()
- Receive control flow of data in blocking mode using HAL_SPDIFRX_ReceiveControlFlow()

Interrupt mode for reception operation

- Receive an amount of data (Data Flow) in non blocking mode using HAL_SPDIFRX_ReceiveDataFlow_IT()
- Receive an amount of data (Control Flow) in non blocking mode using HAL_SPDIFRX_ReceiveControlFlow_IT()
- At reception end of half transfer HAL_SPDIFRX_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxHalfCpltCallback
- At reception end of transfer HAL_SPDIFRX_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxCpltCallback
- In case of transfer Error, HAL_SPDIFRX_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_ErrorCallback

DMA mode for reception operation

- Receive an amount of data (Data Flow) in non blocking mode (DMA) using HAL_SPDIFRX_ReceiveDataFlow_DMA()
- Receive an amount of data (Control Flow) in non blocking mode (DMA) using HAL_SPDIFRX_ReceiveControlFlow_DMA()
- At reception end of half transfer HAL_SPDIFRX_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxHalfCpltCallback
- At reception end of transfer HAL_SPDIFRX_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_RxCpltCallback
- In case of transfer Error, HAL_SPDIFRX_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SPDIFRX_ErrorCallback
- Stop the DMA Transfer using HAL_SPDIFRX_DMAStop()

SPDIFRX HAL driver macros list

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

- __HAL_SPDIFRX_IDLE: Disable the specified SPDIFRX peripheral (IDLE State)
- __HAL_SPDIFRX_SYNC: Enable the synchronization state of the specified SPDIFRX peripheral (SYNC State)
- __HAL_SPDIFRX_RCV: Enable the receive state of the specified SPDIFRX peripheral (RCV State)
- __HAL_SPDIFRX_ENABLE_IT : Enable the specified SPDIFRX interrupts
- __HAL_SPDIFRX_DISABLE_IT : Disable the specified SPDIFRX interrupts

- `__HAL_SPDIFRX_GET_FLAG`: Check whether the specified SPDIFRX flag is set or not.

Note: You can refer to the SPDIFRX HAL driver header file for more useful macros

68.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPDIFRX peripheral:

- User must Implement `HAL_SPDIFRX_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function `HAL_SPDIFRX_Init()` to configure the SPDIFRX peripheral with the selected configuration:
 - Input Selection (IN0, IN1,...)
 - Maximum allowed re-tries during synchronization phase
 - Wait for activity on SPDIF selected input
 - Channel status selection (from channel A or B)
 - Data format (LSB, MSB, ...)
 - Stereo mode
 - User bits masking (PT,C,U,V,...)
- Call the function `HAL_SPDIFRX_DeInit()` to restore the default configuration of the selected SPDIFRXx peripheral.

This section contains the following APIs:

- [`HAL_SPDIFRX_Init`](#)
- [`HAL_SPDIFRX_DeInit`](#)
- [`HAL_SPDIFRX_MspInit`](#)
- [`HAL_SPDIFRX_MspDeInit`](#)
- [`HAL_SPDIFRX_SetDataFormat`](#)

68.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPDIFRX data transfers.

1. There is two mode of transfer:
 - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer start-up. The end of the data processing will be indicated through the dedicated SPDIFRX IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
 - `HAL_SPDIFRX_ReceiveDataFlow()`
 - `HAL_SPDIFRX_ReceiveControlFlow()` (+@) Do not use blocking mode to receive both control and data flow at the same time.
3. No-Blocking mode functions with Interrupt are :
 - `HAL_SPDIFRX_ReceiveControlFlow_IT()`
 - `HAL_SPDIFRX_ReceiveDataFlow_IT()`
4. No-Blocking mode functions with DMA are :
 - `HAL_SPDIFRX_ReceiveControlFlow_DMA()`
 - `HAL_SPDIFRX_ReceiveDataFlow_DMA()`
5. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
 - `HAL_SPDIFRX_RxCpltCallback()`
 - `HAL_SPDIFRX_ErrorCallback()`

This section contains the following APIs:

- [`HAL_SPDIFRX_ReceiveDataFlow`](#)
- [`HAL_SPDIFRX_ReceiveControlFlow`](#)

- `HAL_SPDIFRX_ReceiveDataFlow_IT`
- `HAL_SPDIFRX_ReceiveControlFlow_IT`
- `HAL_SPDIFRX_ReceiveDataFlow_DMA`
- `HAL_SPDIFRX_ReceiveControlFlow_DMA`
- `HAL_SPDIFRX_DMAShort`
- `HAL_SPDIFRX_IRQHandler`
- `HAL_SPDIFRX_RxHalfCpltCallback`
- `HAL_SPDIFRX_RxCpltCallback`
- `HAL_SPDIFRX_CxHalfCpltCallback`
- `HAL_SPDIFRX_CxCpltCallback`
- `HAL_SPDIFRX_ErrorCallback`

68.2.4 Peripheral State and Errors functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- `HAL_SPDIFRX_GetState`
- `HAL_SPDIFRX_GetError`

68.2.5 Detailed description of functions

`HAL_SPDIFRX_Init`

Function name

`HAL_StatusTypeDef HAL_SPDIFRX_Init (SPDIFRX_HandleTypeDef * hspdif)`

Function description

Initializes the SPDIFRX according to the specified parameters in the SPDIFRX_InitTypeDef and create the associated handle.

Parameters

- `hspdif`: SPDIFRX handle

Return values

- `HAL`: status

`HAL_SPDIFRX_DeInit`

Function name

`HAL_StatusTypeDef HAL_SPDIFRX_DeInit (SPDIFRX_HandleTypeDef * hspdif)`

Function description

Deinitializes the SPDIFRX peripheral.

Parameters

- `hspdif`: SPDIFRX handle

Return values

- `HAL`: status

`HAL_SPDIFRX_MspInit`

Function name

`void HAL_SPDIFRX_MspInit (SPDIFRX_HandleTypeDef * hspdif)`

Function description

SPDIFRX MSP Init.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_MspDeInit

Function name

void HAL_SPDIFRX_MspDeInit (SPDIFRX_HandleTypeDef * hspdif)

Function description

SPDIFRX MSP Delinit.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_SetDataFormat

Function name

**HAL_StatusTypeDef HAL_SPDIFRX_SetDataFormat (SPDIFRX_HandleTypeDef * hspdif,
SPDIFRX_SetDataFormatTypeDef sDataFormat)**

Function description

Sets the SPDIFRX data format according to the specified parameters in the SPDIFRX_InitTypeDef.

Parameters

- **hspdif:** SPDIFRX handle
- **sDataFormat:** SPDIFRX data format

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveDataFlow

Function name

**HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t *
pData, uint16_t Size, uint32_t Timeout)**

Function description

Receives an amount of data (Data Flow) in blocking mode.

Parameters

- **hspdif:** pointer to SPDIFRX_HandleTypeDef structure that contains the configuration information for SPDIFRX module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveControlFlow

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives an amount of data (Control Flow) in blocking mode.

Parameters

- **hspdif:** pointer to a SPDIFRX_HandleTypeDef structure that contains the configuration information for SPDIFRX module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveControlFlow_IT

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow_IT (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data (Control Flow) with Interrupt.

Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data sample (Control Flow) to be received :

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveDataFlow_IT

Function name

HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_IT (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data (Data Flow) in non-blocking mode with Interrupt.

Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be received .

Return values

- **HAL:** status

HAL_SPDIFRX_IRQHandler

Function name

```
void HAL_SPDIFRX_IRQHandler (SPDIFRX_HandleTypeDef * hspdif)
```

Function description

This function handles SPDIFRX interrupt request.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveControlFlow_DMA

Function name

```
HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow_DMA (SPDIFRX_HandleTypeDef * hspdif,  
          uint32_t * pData, uint16_t Size)
```

Function description

Receive an amount of data (Control Flow) with DMA.

Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data (Control Flow) sample to be received :

Return values

- **HAL:** status

HAL_SPDIFRX_ReceiveDataFlow_DMA

Function name

```
HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_DMA (SPDIFRX_HandleTypeDef * hspdif, uint32_t  
          * pData, uint16_t Size)
```

Function description

Receive an amount of data (Data Flow) mode with DMA.

Parameters

- **hspdif:** SPDIFRX handle
- **pData:** a 32-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be received :

Return values

- **HAL:** status

HAL_SPDIFRX_DMASStop

Function name

```
HAL_StatusTypeDef HAL_SPDIFRX_DMASStop (SPDIFRX_HandleTypeDef * hspdif)
```

Function description

stop the audio stream receive from the Media.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_RxHalfCpltCallback

Function name

void HAL_SPDIFRX_RxHalfCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

Rx Transfer (Data flow) half completed callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_RxCpltCallback

Function name

void HAL_SPDIFRX_RxCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

Rx Transfer (Data flow) completed callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_ErrorCallback

Function name

void HAL_SPDIFRX_ErrorCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

SPDIFRX error callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_CxHalfCpltCallback

Function name

void HAL_SPDIFRX_CxHalfCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

Rx (Control flow) Transfer half completed callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_CxCpltCallback

Function name

void HAL_SPDIFRX_CxCpltCallback (SPDIFRX_HandleTypeDef * hspdif)

Function description

Rx Transfer (Control flow) completed callbacks.

Parameters

- **hspdif:** SPDIFRX handle

Return values

- **None:**

HAL_SPDIFRX_GetState

Function name

HAL_SPDIFRX_StateTypeDef HAL_SPDIFRX_GetState (SPDIFRX_HandleTypeDef * hspdif)

Function description

Return the SPDIFRX state.

Parameters

- **hspdif:** : SPDIFRX handle

Return values

- **HAL:** state

HAL_SPDIFRX_GetError

Function name

uint32_t HAL_SPDIFRX_GetError (SPDIFRX_HandleTypeDef * hspdif)

Function description

Return the SPDIFRX error code.

Parameters

- **hspdif:** : SPDIFRX handle

Return values

- **SPDIFRX:** Error Code

68.3 SPDIFRX Firmware driver defines

68.3.1 SPDIFRX

SPDIFRX Channel Status Mask

SPDIFRX_CHANNELSTATUS_OFF

SPDIFRX_CHANNELSTATUS_ON

SPDIFRX Channel Selection

SPDIFRX_CHANNEL_A**SPDIFRX_CHANNEL_B**

SPDIFRX Data Format

SPDIFRX_DATAFORMAT_LSB**SPDIFRX_DATAFORMAT_MSB****SPDIFRX_DATAFORMAT_32BITS**

SPDIFRX Error Code

HAL_SPDIFRX_ERROR_NONE

No error

HAL_SPDIFRX_ERROR_TIMEOUT

Timeout error

HAL_SPDIFRX_ERROR_OVR

OVR error

HAL_SPDIFRX_ERROR_PE

Parity error

HAL_SPDIFRX_ERROR_DMA

DMA transfer error

HAL_SPDIFRX_ERROR_UNKNOWN

Unknown Error error

SPDIFRX Exported Macros

_HAL_SPDIFRX_RESET_HANDLE_STATE

Description:

- Reset SPDIFRX handle state.

Parameters:

- __HANDLE__: SPDIFRX handle.

Return value:

- None

_HAL_SPDIFRX_IDLE

Description:

- Disable the specified SPDIFRX peripheral (IDLE State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_SYNC

Description:

- Enable the specified SPDIFRX peripheral (SYNC State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_RCV

Description:

- Enable the specified SPDIFRX peripheral (RCV State).

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.

Return value:

- None

__HAL_SPDIFRX_ENABLE_IT

Description:

- Enable or disable the specified SPDIFRX interrupts.

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - SPDIFRX_IT_RXNE
 - SPDIFRX_IT_CSRNE
 - SPDIFRX_IT_PERRIE
 - SPDIFRX_IT_OVRIE
 - SPDIFRX_IT_SBLKIE
 - SPDIFRX_IT_SYNCDIE
 - SPDIFRX_IT_IFEIE

Return value:

- None

__HAL_SPDIFRX_DISABLE_IT

__HAL_SPDIFRX_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __INTERRUPT__: specifies the SPDIFRX interrupt source to check. This parameter can be one of the following values:
 - SPDIFRX_IT_RXNE
 - SPDIFRX_IT_CSRNE
 - SPDIFRX_IT_PERRIE
 - SPDIFRX_IT_OVRIE
 - SPDIFRX_IT_SBLKIE

- SPDIFRX_IT_SYNCDIE
- SPDIFRX_IT_IFEIE

Return value:

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

[_HAL_SPDIFRX_GET_FLAG](#)**Description:**

- Checks whether the specified SPDIFRX flag is set or not.

Parameters:

- __HANDLE__: specifies the SPDIFRX Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - SPDIFRX_FLAG_RXNE
 - SPDIFRX_FLAG_CSRNE
 - SPDIFRX_FLAG_PERR
 - SPDIFRX_FLAG_OVR
 - SPDIFRX_FLAG_SBD
 - SPDIFRX_FLAG_SYNCDE
 - SPDIFRX_FLAG_FERR
 - SPDIFRX_FLAG_SERR
 - SPDIFRX_FLAG_TERR

Return value:

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

[_HAL_SPDIFRX_CLEAR_IT](#)**Description:**

- Clears the specified SPDIFRX SR flag, in setting the proper IFCR register bit.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
 - SPDIFRX_FLAG_PERR
 - SPDIFRX_FLAG_OVR
 - SPDIFRX_SR_SBD
 - SPDIFRX_SR_SYNCDE

Return value:

- None

SPDIFRX Flags Definition[SPDIFRX_FLAG_RXNE](#)[SPDIFRX_FLAG_CSRNE](#)[SPDIFRX_FLAG_PERR](#)[SPDIFRX_FLAG_OVR](#)[SPDIFRX_FLAG_SBD](#)

SPDIFRX_FLAG_SYNCND

SPDIFRX_FLAG_FERR

SPDIFRX_FLAG_SERR

SPDIFRX_FLAG_TERR

SPDIFRX Input Selection

SPDIFRX_INPUT_IN0

SPDIFRX_INPUT_IN1

SPDIFRX_INPUT_IN2

SPDIFRX_INPUT_IN3

SPDIFRX Interrupts Definition

SPDIFRX_IT_RXNE

SPDIFRX_IT_CSRNE

SPDIFRX_IT_PERRIE

SPDIFRX_IT_OVRIE

SPDIFRX_IT_SBLKIE

SPDIFRX_IT_SYNCDIE

SPDIFRX_IT_IFEIE

SPDIFRX Maximum Retries

SPDIFRX_MAXRETRIES_NONE

SPDIFRX_MAXRETRIES_3

SPDIFRX_MAXRETRIES_15

SPDIFRX_MAXRETRIES_63

SPDIFRX Parity Error Mask

SPDIFRX_PARITYERRORMASK_OFF

SPDIFRX_PARITYERRORMASK_ON

SPDIFRX Preamble Type Mask

SPDIFRX_PREAMBLETYPEMASK_OFF

SPDIFRX_PREAMBLETYPEMASK_ON

SPDIFRX State

SPDIFRX_STATE_IDLE

SPDIFRX_STATE_SYNC

SPDIFRX_STATE_RCV

SPDIFRX Stereo Mode

SPDIFRX_STEREOMODE_DISABLE

SPDIFRX_STEREOMODE_ENABLE

SPDIFRX Validity Mask

SPDIFRX_VALIDITYMASK_OFF

SPDIFRX_VALIDITYMASK_ON

SPDIFRX Wait For Activity

SPDIFRX_WAITFORACTIVITY_OFF

SPDIFRX_WAITFORACTIVITY_ON

69 HAL SPI Generic Driver

69.1 SPI Firmware driver registers structures

69.1.1 SPI_InitTypeDef

Data Fields

- *uint32_t Mode*
- *uint32_t Direction*
- *uint32_t DataSize*
- *uint32_t CLKPolarity*
- *uint32_t CLKPhase*
- *uint32_t NSS*
- *uint32_t BaudRatePrescaler*
- *uint32_t FirstBit*
- *uint32_t TIMode*
- *uint32_t CRCCalculation*
- *uint32_t CRCPolynomial*
- *uint32_t CRCLength*
- *uint32_t NSSPMode*
- *uint32_t NSSPolarity*
- *uint32_t FifoThreshold*
- *uint32_t TxCRCInitializationPattern*
- *uint32_t RxCRCInitializationPattern*
- *uint32_t MasterSSIdleness*
- *uint32_t MasterInterDataIdleness*
- *uint32_t MasterReceiverAutoSusp*
- *uint32_t MasterKeepIOState*
- *uint32_t IOSwap*

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
Specifies the SPI operating mode. This parameter can be a value of **SPI Mode**
- ***uint32_t SPI_InitTypeDef::Direction***
Specifies the SPI bidirectional mode state. This parameter can be a value of **SPI Direction Mode**
- ***uint32_t SPI_InitTypeDef::DataSize***
Specifies the SPI data size. This parameter can be a value of **SPI Data Size**
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
Specifies the serial clock steady state. This parameter can be a value of **SPI Clock Polarity**
- ***uint32_t SPI_InitTypeDef::CLKPhase***
Specifies the clock active edge for the bit capture. This parameter can be a value of **SPI Clock Phase**
- ***uint32_t SPI_InitTypeDef::NSS***
Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of **SPI Slave Select Management**

- **`uint32_t SPI_InitTypeDef::BaudRatePrescaler`**

Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of **SPI BaudRate Prescaler**

Note:

- The communication clock is derived from the master clock. The slave clock does not need to be set.

- **`uint32_t SPI_InitTypeDef::FirstBit`**

Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of **SPI MSB LSB Transmission**

- **`uint32_t SPI_InitTypeDef::TIMode`**

Specifies if the TI mode is enabled or not. This parameter can be a value of **SPI TI Mode**

- **`uint32_t SPI_InitTypeDef::CRCCalculation`**

Specifies if the CRC calculation is enabled or not. This parameter can be a value of **SPI CRC Calculation**

- **`uint32_t SPI_InitTypeDef::CRCPolynomial`**

Specifies the polynomial used for the CRC calculation. This parameter must be an odd number between Min_Data = 0 and Max_Data = 65535

- **`uint32_t SPI_InitTypeDef::CRCLength`**

Specifies the CRC Length used for the CRC calculation. CRC Length is only used with Data8 and Data16, not other data size This parameter can be a value of **SPI CRC Length**

- **`uint32_t SPI_InitTypeDef::NSSPMode`**

Specifies whether the NSSP signal is enabled or not . This parameter can be a value of **SPI NSS Pulse Mode** This mode is activated by the NSSP bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx_CR1 CPHA = 0, CPOL setting is ignored).

- **`uint32_t SPI_InitTypeDef::NSSPolarity`**

Specifies which level of SS input/output external signal (present on SS pin) is considered as active one. This parameter can be a value of **SPI NSS Polarity**

- **`uint32_t SPI_InitTypeDef::FifoThreshold`**

Specifies the FIFO threshold level. This parameter can be a value of **SPI Fifo Threshold**

- **`uint32_t SPI_InitTypeDef::TxCRCInitializationPattern`**

Specifies the transmitter CRC initialization Pattern used for the CRC calculation. This parameter can be a value of **SPI CRC Calculation Initialization Pattern**

- **`uint32_t SPI_InitTypeDef::RxCRCInitializationPattern`**

Specifies the receiver CRC initialization Pattern used for the CRC calculation. This parameter can be a value of **SPI CRC Calculation Initialization Pattern**

- **`uint32_t SPI_InitTypeDef::MasterSSIdleness`**

Specifies an extra delay, expressed in number of SPI clock cycle periods, inserted additionally between active edge of SS and first data transaction start in master mode. This parameter can be a value of **SPI Master SS Idleness**

- **`uint32_t SPI_InitTypeDef::MasterInterDataIdleness`**

Specifies minimum time delay (expressed in SPI clock cycles periods) inserted between two consecutive data frames in master mode This parameter can be a value of **SPI Master Inter-Data Idleness**

- **`uint32_t SPI_InitTypeDef::MasterReceiverAutoSusp`**

Control continuous SPI transfer in master receiver mode and automatic management in order to avoid overrun condition. This parameter can be a value of **SPI Master Receiver AutoSuspend**

- **`uint32_t SPI_InitTypeDef::MasterKeepIOState`**

Control of Alternate function GPIOs state This parameter can be a value of **Keep IO State**

- **`uint32_t SPI_InitTypeDef::IOSwap`**

Invert MISO/MOSI alternate functions This parameter can be a value of **Control SPI IO Swap**

69.1.2 **`__SPI_HandleTypeDef`**

Data Fields

- **`SPI_TypeDef * Instance`**
- **`SPI_InitTypeDef Init`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferSize`**
- **`_IO uint16_t TxXferCount`**
- **`uint8_t * pRxBuffPtr`**
- **`uint16_t RxXferSize`**
- **`_IO uint16_t RxXferCount`**
- **`uint32_t CRCSize`**
- **`void(* RxISR`**
- **`void(* TxISR`**
- **`DMA_HandleTypeDef * hdmatx`**
- **`DMA_HandleTypeDef * hdmarx`**
- **`HAL_LockTypeDef Lock`**
- **`_IO HAL_SPI_StateTypeDef State`**
- **`_IO uint32_t ErrorCode`**

Field Documentation

- **`SPI_TypeDef* __SPI_HandleTypeDef::Instance`**
SPI registers base address
- **`SPI_InitTypeDef __SPI_HandleTypeDef::Init`**
SPI communication parameters
- **`uint8_t* __SPI_HandleTypeDef::pTxBuffPtr`**
Pointer to SPI Tx transfer Buffer
- **`uint16_t __SPI_HandleTypeDef::TxXferSize`**
SPI Tx Transfer size
- **`_IO uint16_t __SPI_HandleTypeDef::TxXferCount`**
SPI Tx Transfer Counter
- **`uint8_t* __SPI_HandleTypeDef::pRxBuffPtr`**
Pointer to SPI Rx transfer Buffer
- **`uint16_t __SPI_HandleTypeDef::RxXferSize`**
SPI Rx Transfer size
- **`_IO uint16_t __SPI_HandleTypeDef::RxXferCount`**
SPI Rx Transfer Counter
- **`uint32_t __SPI_HandleTypeDef::CRCSize`**
SPI CRC size used for the transfer
- **`void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)`**
function pointer on Rx ISR

- `void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)`
function pointer on Tx ISR
- `DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx`
SPI Tx DMA Handle parameters
- `DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx`
SPI Rx DMA Handle parameters
- `HAL_LockTypeDef __SPI_HandleTypeDef::Lock`
Locking object
- `_IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State`
SPI communication state
- `_IO uint32_t __SPI_HandleTypeDef::ErrorCode`
SPI Error code

69.2 SPI Firmware driver API description

69.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
 - a. Enable the SPIx interface clock
 - b. SPI pins configuration
 - Enable the clock for the SPI GPIOs
 - Configure these SPI pins as alternate function push-pull
 - c. NVIC configuration if you need to use interrupt process or DMA process
 - Configure the SPIx interrupt priority
 - Enable the NVIC SPI IRQ handle
 - d. DMA Configuration if you need to use DMA process
 - Declare a DMA_HandleTypeDef handle structure for the transmit or receive Stream/Channel
 - Enable the DMAx clock
 - Configure the DMA handle parameters
 - Configure the DMA Tx or Rx Stream/Channel
 - Associate the initialized hdma_tx handle to the hspi DMA Tx or Rx handle
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream/Channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Circular mode restriction:

- The DMA circular mode cannot be used when the SPI is configured in these modes:
 - Master 2Lines RxOnly
 - Master 1Line Rx
- The CRC feature is not managed when the DMA circular mode is enabled

- The functions HAL_SPI_DMAPause() / HAL_SPI_DMAResume() are not supported. Return always HAL_ERROR with ErrorCode set to HAL_SPI_ERROR_NOT_SUPPORTED. Those functions are maintained for backward compatibility reasons.

69.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
 - Mode
 - Direction
 - Data Size
 - Clock Polarity and Phase
 - NSS Management
 - BaudRate Prescaler
 - FirstBit
 - TIMode
 - CRC Calculation
 - CRC Polynomial if CRC enabled
 - CRC Length, used only with Data8 and Data16
 - FIFO reception threshold
 - FIFO transmission threshold
- Call the function HAL_SPI_DelInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- [**• HAL_SPI_Init**](#)
- [**• HAL_SPI_DelInit**](#)
- [**• HAL_SPI_MspInit**](#)
- [**• HAL_SPI_MspDelInit**](#)

69.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPI data transfers.

The SPI supports master and slave mode :

- There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process. The HAL_SPI_ErrorCallback() user callback will be executed when a communication error is detected
- APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- [**• HAL_SPI_Transmit**](#)
- [**• HAL_SPI_Receive**](#)
- [**• HAL_SPI_TransmitReceive**](#)
- [**• HAL_SPI_Transmit_IT**](#)
- [**• HAL_SPI_Receive_IT**](#)

- [*HAL_SPI_TransmitReceive_IT*](#)
- [*HAL_SPI_Transmit_DMA*](#)
- [*HAL_SPI_Receive_DMA*](#)
- [*HAL_SPI_TransmitReceive_DMA*](#)
- [*HAL_SPI_Abort*](#)
- [*HAL_SPI_Abort_IT*](#)
- [*HAL_SPI_DMAPause*](#)
- [*HAL_SPI_DMAResume*](#)
- [*HAL_SPI_DMAStop*](#)
- [*HAL_SPI_IRQHandler*](#)
- [*HAL_SPI_TxCpltCallback*](#)
- [*HAL_SPI_RxCpltCallback*](#)
- [*HAL_SPI_TxRxCpltCallback*](#)
- [*HAL_SPI_TxHalfCpltCallback*](#)
- [*HAL_SPI_RxHalfCpltCallback*](#)
- [*HAL_SPI_TxRxHalfCpltCallback*](#)
- [*HAL_SPI_ErrorCallback*](#)
- [*HAL_SPI_AbortCpltCallback*](#)

69.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to control the SPI.

- [*HAL_SPI_GetState\(\)*](#) API can be helpful to check in run-time the state of the SPI peripheral
- [*HAL_SPI_GetError\(\)*](#) check in run-time Errors occurring during communication

This section contains the following APIs:

- [*HAL_SPI_GetState*](#)
- [*HAL_SPI_GetError*](#)

69.2.5 Detailed description of functions

[**HAL_SPI_Init**](#)

Function name

HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI according to the specified parameters in the SPI_InitTypeDef and initialize the associated handle.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

[**HAL_SPI_DeInit**](#)

Function name

HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI peripheral.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_MspInit

Function name

```
void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)
```

Function description

Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_MspDeInit

Function name

```
void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)
```

Function description

De-Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_Transmit

Function name

```
HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Receive

Function name

```
HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Receive an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_TransmitReceive

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Transmit and Receive an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent and received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_TransmitReceive_IT

Function name

```
HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t  
* pRxData, uint16_t Size)
```

Function description

Transmit and Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent and received

Return values

- **HAL:** status

HAL_SPI_Transmit_DMA

Function name

```
HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
```

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_DMA

Function name

```
HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
```

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pData Length must be Size + 1.

HAL_SPI_TransmitReceive_DMA

Function name

```
HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData,  
uint8_t * pRxData, uint16_t Size)
```

Function description

Transmit and Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pRxData Length must be Size + 1

HAL_SPI_DMAPause

Function name

```
HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)
```

Function description

Pause the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_DMAResume

Function name

```
HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)
```

Function description

Resume the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_DMAMainStop

Function name

`HAL_StatusTypeDef HAL_SPI_DMAMainStop (SPI_HandleTypeDef * hspi)`

Function description

Stop the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_Abort

Function name

`HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)`

Function description

Abort ongoing transfer (blocking mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode.
- This procedure performs following operations : + Disable SPI Interrupts (depending of transfer direction) + Disable the DMA transfer in the peripheral register (if enabled) + Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode) + Set handle State to READY.
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SPI_Abort_IT

Function name

`HAL_StatusTypeDef HAL_SPI_Abort_IT (SPI_HandleTypeDef * hspi)`

Function description

Abort ongoing transfer (Interrupt mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode.
- This procedure performs following operations : + Disable SPI Interrupts (depending of transfer direction) + Disable the DMA transfer in the peripheral register (if enabled) + Abort DMA transfer by calling

- HAL_DMA_Abort_IT (in case of transfer in DMA mode) + Set handle State to READY + At abort completion, call user abort complete callback.
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SPI_IRQHandler

Function name

void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)

Function description

Handle SPI interrupt request.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **None:**

HAL_SPI_TxCpltCallback

Function name

void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxCpltCallback

Function name

void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxCpltCallback

Function name

void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxHalfCpltCallback

Function name

void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxHalfCpltCallback

Function name

void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxHalfCpltCallback

Function name

void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Half Transfer callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_ErrorCallback

Function name

void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)

Function description

SPI error callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_AbortCpltCallback

Function name

void HAL_SPI_AbortCpltCallback (SPI_HandleTypeDef * hspi)

Function description

SPI Abort Complete callback.

Parameters

- **hspi:** SPI handle.

Return values

- **None:**

HAL_SPI_GetState

Function name

HAL_StatusTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)

Function description

Return the SPI handle state.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** state

HAL_SPI_GetError

Function name

uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)

Function description

Return the SPI error code.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** error code in bitmap format

69.3 SPI Firmware driver defines

69.3.1 SPI

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4
SPI_BAUDRATEPRESCALER_8
SPI_BAUDRATEPRESCALER_16
SPI_BAUDRATEPRESCALER_32
SPI_BAUDRATEPRESCALER_64
SPI_BAUDRATEPRESCALER_128
SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI CRC Calculation Initialization Pattern

SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN

SPI_CRC_INITIALIZATION_ALL_ONE_PATTERN

SPI CRC Length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_4BIT

SPI_CRC_LENGTH_5BIT

SPI_CRC_LENGTH_6BIT

SPI_CRC_LENGTH_7BIT

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_9BIT

SPI_CRC_LENGTH_10BIT

SPI_CRC_LENGTH_11BIT

SPI_CRC_LENGTH_12BIT

SPI_CRC_LENGTH_13BIT

SPI_CRC_LENGTH_14BIT

SPI_CRC_LENGTH_15BIT

SPI_CRC_LENGTH_16BIT

SPI_CRC_LENGTH_17BIT

SPI_CRC_LENGTH_18BIT

SPI_CRC_LENGTH_19BIT

SPI_CRC_LENGTH_20BIT

SPI_CRC_LENGTH_21BIT

SPI_CRC_LENGTH_22BIT

SPI_CRC_LENGTH_23BIT

SPI_CRC_LENGTH_24BIT

SPI_CRC_LENGTH_25BIT

SPI_CRC_LENGTH_26BIT

SPI_CRC_LENGTH_27BIT

SPI_CRC_LENGTH_28BIT

SPI_CRC_LENGTH_29BIT

SPI_CRC_LENGTH_30BIT

SPI_CRC_LENGTH_31BIT

SPI_CRC_LENGTH_32BIT

SPI Data Size

SPI_DATASIZE_4BIT

SPI_DATASIZE_5BIT

SPI_DATASIZE_6BIT

SPI_DATASIZE_7BIT

SPI_DATASIZE_8BIT

SPI_DATASIZE_9BIT

SPI_DATASIZE_10BIT

`SPI_DATASIZE_11BIT`

`SPI_DATASIZE_12BIT`

`SPI_DATASIZE_13BIT`

`SPI_DATASIZE_14BIT`

`SPI_DATASIZE_15BIT`

`SPI_DATASIZE_16BIT`

`SPI_DATASIZE_17BIT`

`SPI_DATASIZE_18BIT`

`SPI_DATASIZE_19BIT`

`SPI_DATASIZE_20BIT`

`SPI_DATASIZE_21BIT`

`SPI_DATASIZE_22BIT`

`SPI_DATASIZE_23BIT`

`SPI_DATASIZE_24BIT`

`SPI_DATASIZE_25BIT`

`SPI_DATASIZE_26BIT`

`SPI_DATASIZE_27BIT`

`SPI_DATASIZE_28BIT`

`SPI_DATASIZE_29BIT`

`SPI_DATASIZE_30BIT`

`SPI_DATASIZE_31BIT`

`SPI_DATASIZE_32BIT`

SPI Direction Mode

`SPI_DIRECTION_2LINES`

`SPI_DIRECTION_2LINES_TXONLY`

`SPI_DIRECTION_2LINES_RXONLY`

`SPI_DIRECTION_1LINE`

SPI Error Codes

HAL_SPI_ERROR_NONE

No error

HAL_SPI_ERROR_MODF

MODF error

HAL_SPI_ERROR_CRC

CRC error

HAL_SPI_ERROR_OVR

OVR error

HAL_SPI_ERROR_FRE

FRE error

HAL_SPI_ERROR_DMA

DMA transfer error

HAL_SPI_ERROR_FLAG

Error on RXNE/TXE/BSY/FTLVL/FRLVL Flag

HAL_SPI_ERROR_ABORT

Error during SPI Abort procedure

HAL_SPI_ERROR_UDR

Underrun error

HAL_SPI_ERROR_TIMEOUT

Timeout error

HAL_SPI_ERROR_UNKNOW

Unknow error

HAL_SPI_ERROR_NOT_SUPPORTED

Requested operation not supported

SPI Exported Macros**__HAL_SPI_RESET_HANDLE_STATE****Description:**

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT**Description:**

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.

- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - SPI_IT_TXE: Tx buffer empty interrupt enable
 - SPI_IT_RXNE: RX buffer not empty interrupt enable
 - SPI_IT_ERR: Error interrupt enable

Return value:

- None

[__HAL_SPI_DISABLE_IT](#)**Description:**

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - SPI_IT_TXE: Tx buffer empty interrupt enable
 - SPI_IT_RXNE: RX buffer not empty interrupt enable
 - SPI_IT_ERR: Error interrupt enable

Return value:

- None

[__HAL_SPI_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
 - SPI_IT_TXE: Tx buffer empty interrupt enable
 - SPI_IT_RXNE: RX buffer not empty interrupt enable
 - SPI_IT_ERR: Error interrupt enable

Return value:

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

[__HAL_SPI_GET_FLAG](#)**Description:**

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

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - SPI_FLAG_TXE : Tx buffer empty flag
 - SPI_FLAG_RXNE : Rx buffer not empty flag
 - SPI_FLAG_UDR : Underrun flag
 - SPI_FLAG_OVR : Overrun flag
 - SPI_FLAG_FRE : TI mode frame format error flag

- SPI_FLAG_CRCERR: CRC error flag
- SPI_FLAG_MODF : Mode fault flag
- SPI_FLAG_FRLVL : fifo reception level
- SPI_FLAG_RXWNE : RxFIFO Word Not Empty
- SPI_FLAG_TXTF : Transmission Transfer Filled flag
- SPI_FLAG_EOT : fifo transmission complete

Return value:

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

[__HAL_SPI_CLEAR_CRCERRFLAG](#)**Description:**

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_CLEAR_MODFFLAG](#)**Description:**

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_CLEAR_OVRFAG](#)**Description:**

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_CLEAR_FREFLAG](#)**Description:**

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_CLEAR_UDRFLAG](#)**Description:**

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_CLEAR_EOTFLAG](#)**Description:**

- Clear the SPI EOT pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_CLEAR_TXTFFLAG](#)**Description:**

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_CLEAR_SUSPFLAG](#)**Description:**

- Clear the SPI SUSP pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_ENABLE](#)**Description:**

- Enable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

[__HAL_SPI_DISABLE](#)**Description:**

- Disable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

SPI Fifo Threshold[SPI_FIFO_THRESHOLD_01DATA](#)

SPI_FIFO_THRESHOLD_02DATA

SPI_FIFO_THRESHOLD_03DATA

SPI_FIFO_THRESHOLD_04DATA

SPI_FIFO_THRESHOLD_05DATA

SPI_FIFO_THRESHOLD_06DATA

SPI_FIFO_THRESHOLD_07DATA

SPI_FIFO_THRESHOLD_08DATA

SPI_FIFO_THRESHOLD_09DATA

SPI_FIFO_THRESHOLD_10DATA

SPI_FIFO_THRESHOLD_11DATA

SPI_FIFO_THRESHOLD_12DATA

SPI_FIFO_THRESHOLD_13DATA

SPI_FIFO_THRESHOLD_14DATA

SPI_FIFO_THRESHOLD_15DATA

SPI_FIFO_THRESHOLD_16DATA

SPI FIFO Type

SPI_LOWEND_FIFO_SIZE

SPI_HIGHEND_FIFO_SIZE

SPI Flags Definition

SPI_FLAG_TXE

SPI_FLAG_RXNE

SPI_FLAG_UDR

SPI_FLAG_OVR

SPI_FLAG_FRE

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_FRLVL

SPI_FLAG_RXWNE

SPI_FLAG_TXTF

`SPI_FLAG_EOT`

`SPI_FLAG_SUSP`

SPI Interrupt Definition

`SPI_IT_TXE`

`SPI_IT_RXNE`

`SPI_IT_EOT`

`SPI_IT_TXTF`

`SPI_IT_UDR`

`SPI_IT_OVR`

`SPI_IT_FRE`

`SPI_IT_MODF`

`SPI_IT_ERR`

Control SPI IO Swap

`SPI_IO_SWAP_DISABLE`

`SPI_IO_SWAP_ENABLE`

SPI Master Inter-Data Idleness

`SPI_MASTER_INTERDATA_IDLENESS_00CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_01CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_02CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_03CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_04CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_05CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_06CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_07CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_08CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_09CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_10CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_11CYCLE`

`SPI_MASTER_INTERDATA_IDLENESS_12CYCLE`

SPI_MASTER_INTERDATA_IDLENESS_13CYCLE

SPI_MASTER_INTERDATA_IDLENESS_14CYCLE

SPI_MASTER_INTERDATA_IDLENESS_15CYCLE

Keep IO State

SPI_MASTER_KEEP_IO_STATE_DISABLE

SPI_MASTER_KEEP_IO_STATE_ENABLE

SPI Master Receiver AutoSuspend

SPI_MASTER_RX_AUTOSUSP_DISABLE

SPI_MASTER_RX_AUTOSUSP_ENABLE

SPI Master SS Idleness

SPI_MASTER_SS_IDLENESS_00CYCLE

SPI_MASTER_SS_IDLENESS_01CYCLE

SPI_MASTER_SS_IDLENESS_02CYCLE

SPI_MASTER_SS_IDLENESS_03CYCLE

SPI_MASTER_SS_IDLENESS_04CYCLE

SPI_MASTER_SS_IDLENESS_05CYCLE

SPI_MASTER_SS_IDLENESS_06CYCLE

SPI_MASTER_SS_IDLENESS_07CYCLE

SPI_MASTER_SS_IDLENESS_08CYCLE

SPI_MASTER_SS_IDLENESS_09CYCLE

SPI_MASTER_SS_IDLENESS_10CYCLE

SPI_MASTER_SS_IDLENESS_11CYCLE

SPI_MASTER_SS_IDLENESS_12CYCLE

SPI_MASTER_SS_IDLENESS_13CYCLE

SPI_MASTER_SS_IDLENESS_14CYCLE

SPI_MASTER_SS_IDLENESS_15CYCLE

SPI Mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

SPI MSB LSB Transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

SPI NSS Pulse Mode

SPI_NSS_PULSE_DISABLE

SPI_NSS_PULSE_ENABLE

SPI NSS Polarity

SPI_NSS_POLARITY_LOW

SPI_NSS_POLARITY_HIGH

SPI Reception FIFO Status Level

SPI_FRLVL_EMPTY

SPI_FRLVL_QUARTER_FULL

SPI_FRLVL_HALF_FULL

SPI_FRLVL_FULL

SPI Slave Select Management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI Mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

SPI Underrun Behaviour

SPI_UNDERRUN_BEHAV_REGISTER_PATTERN

SPI_UNDERRUN_BEHAV_LAST_RECEIVED

SPI_UNDERRUN_BEHAV_LAST_TRANSMITTED

SPI Underrun Detection

SPI_UNDERRUN_DETECT_BEGIN_DATA_FRAME

SPI_UNDERRUN_DETECT_END_DATA_FRAME

SPI_UNDERRUN_DETECT_BEGIN_ACTIVE_NSS

70 HAL SPI Extension Driver

70.1 SPIEx Firmware driver API description

70.1.1 IO operation functions

This subsection provides a set of extended functions to manage the SPI data transfers.

1. SPIEx function:

- HAL_SPIEx_FlushRxFifo()
- HAL_SPIEx_FlushRxFifo()
- HAL_SPIEx_EnableLockConfiguration()
- HAL_SPIEx_ConfigureUnderrun()

This section contains the following APIs:

- [**HAL_SPIEx_FlushRxFifo**](#)
- [**HAL_SPIEx_EnableLockConfiguration**](#)
- [**HAL_SPIEx_ConfigureUnderrun**](#)

70.1.2 Detailed description of functions

HAL_SPIEx_FlushRxFifo

Function name

HAL_StatusTypeDef HAL_SPIEx_FlushRxFifo (SPI_HandleTypeDef * hspi)

Function description

Flush the RX fifo.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL:** status

HAL_SPIEx_EnableLockConfiguration

Function name

HAL_StatusTypeDef HAL_SPIEx_EnableLockConfiguration (SPI_HandleTypeDef * hspi)

Function description

Enable the Lock for the AF configuration of associated IOs and write protect the Content of Configuration register 2 when SPI is enabled.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPIEx_ConfigureUnderrun

Function name

```
HAL_StatusTypeDef HAL_SPIEx_ConfigureUnderrun (SPI_HandleTypeDef * hspi, uint32_t  
UnderrunDetection, uint32_t UnderrunBehaviour)
```

Function description

Configure the UNDERRUN condition and behavior of slave transmitter.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **UnderrunDetection:** : Detection of underrun condition at slave transmitter This parameter can be a value of SPI Underrun Detection.
- **UnderrunBehaviour:** : Behavior of slave transmitter at underrun condition This parameter can be a value of SPI Underrun Behaviour.

Return values

- **None:**

71 HAL SRAM Generic Driver

71.1 SRAM Firmware driver registers structures

71.1.1 SRAM_HandleTypeDef

Data Fields

- `FMC_NORSRAM_TypeDef * Instance`
- `FMC_NORSRAM_EXTENDED_TypeDef * Extended`
- `FMC_NORSRAM_InitTypeDef Init`
- `HAL_LockTypeDef Lock`
- `__IO HAL_SRAM_StateTypeDef State`
- `MDMA_HandleTypeDef * hmdma`

Field Documentation

- **`FMC_NORSRAM_TypeDef* SRAM_HandleTypeDef::Instance`**
Register base address
- **`FMC_NORSRAM_EXTENDED_TypeDef* SRAM_HandleTypeDef::Extended`**
Extended mode register base address
- **`FMC_NORSRAM_InitTypeDef SRAM_HandleTypeDef::Init`**
SRAM device control configuration parameters
- **`HAL_LockTypeDef SRAM_HandleTypeDef::Lock`**
SRAM locking object
- **`__IO HAL_SRAM_StateTypeDef SRAM_HandleTypeDef::State`**
SRAM device access state
- **`MDMA_HandleTypeDef* SRAM_HandleTypeDef::hmdma`**
Pointer DMA handler

71.2 SRAM Firmware driver API description

71.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example: SRAM_HandleTypeDef hsramp; and:
 - Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
 - Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
 - Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC_NORSRAM_TimingTypeDef Timing and FMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.

3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
 - a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()
 - b. Control register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Init()
 - c. Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Timing_Init()
 - d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Extended_Timing_Init()
 - e. Enable the SRAM device using the macro __FMC_NORSRAM_ENABLE()
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
 - HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
 - HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable() / HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

71.2.2

SRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

This section contains the following APIs:

- [HAL_SRAM_Init](#)
- [HAL_SRAM_DelInit](#)
- [HAL_SRAM_MspInit](#)
- [HAL_SRAM_MspDelInit](#)
- [HAL_SRAM_DMA_XferCpltCallback](#)
- [HAL_SRAM_DMA_XferErrorCallback](#)

71.2.3

SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

This section contains the following APIs:

- [HAL_SRAM_Read_8b](#)
- [HAL_SRAM_Write_8b](#)
- [HAL_SRAM_Read_16b](#)
- [HAL_SRAM_Write_16b](#)
- [HAL_SRAM_Read_32b](#)
- [HAL_SRAM_Write_32b](#)
- [HAL_SRAM_Read_DMA](#)
- [HAL_SRAM_Write_DMA](#)
- [HAL_SRAM_DMA_XferCpltCallback](#)
- [HAL_SRAM_DMA_XferErrorCallback](#)

71.2.4

SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

This section contains the following APIs:

- [HAL_SRAM_WriteOperation_Enable](#)
- [HAL_SRAM_WriteOperation_Disable](#)

71.2.5 SRAM State functions

This subsection permits to get in run-time the status of the SRAM controller and the data flow.

This section contains the following APIs:

- [**HAL_SRAM_GetState**](#)

71.2.6 Detailed description of functions

HAL_SRAM_Init

Function name

HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Performs the SRAM device initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **Timing:** Pointer to SRAM control timing structure
- **ExtTiming:** Pointer to SRAM extended mode timing structure

Return values

- **HAL:** status

HAL_SRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)

Function description

Performs the SRAM device De-initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_MspInit

Function name

void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP Init.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_MspDeInit

Function name

```
void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)
```

Function description

SRAM MSP DeInit.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_Read_8b

Function name

```
HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)
```

Function description

Reads 8-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_8b

Function name

```
HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)
```

Function description

Writes 8-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_16b

Function name

```
HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsramp, uint32_t * pAddress,  
uint16_t * pDstBuffer, uint32_t BufferSize)
```

Function description

Reads 16-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_16b

Function name

```
HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsramp, uint32_t * pAddress,  
uint16_t * pSrcBuffer, uint32_t BufferSize)
```

Function description

Writes 16-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_32b

Function name

```
HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsramp, uint32_t * pAddress,  
uint32_t * pDstBuffer, uint32_t BufferSize)
```

Function description

Reads 32-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsramp, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 32-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsramp, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads a Words data from the SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsramp, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes a Words data buffer to SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_DMA_XferCpltCallback

Function name

void HAL_SRAM_DMA_XferCpltCallback (MDMA_HandleTypeDef * hmdma)

Function description

DMA transfer complete callback.

Parameters

- **hmdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_DMA_XferErrorCallback

Function name

void HAL_SRAM_DMA_XferErrorCallback (MDMA_HandleTypeDef * hmdma)

Function description

DMA transfer complete error callback.

Parameters

- **hmdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)

Function description

Enables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)

Function description

Disables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_GetState

Function name

HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)

Function description

Returns the SRAM controller state.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** state

71.3 SRAM Firmware driver defines

71.3.1 SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE

Description:

- Reset SRAM handle state.

Parameters:

- **__HANDLE__:** SRAM handle

Return value:

- None

72 HAL SWPMI Generic Driver

72.1 SWPMI Firmware driver registers structures

72.1.1 SWPMI_InitTypeDef

Data Fields

- *uint32_t VoltageClass*
- *uint32_t BitRate*
- *uint32_t TxBufferingMode*
- *uint32_t RxBufferingMode*

Field Documentation

- *uint32_t SWPMI_InitTypeDef::VoltageClass*

Specifies the SWP Voltage Class. This parameter can be a value of **SWPMI Voltage Class**

- *uint32_t SWPMI_InitTypeDef::BitRate*

Specifies the SWPMI Bitrate. This parameter must be a number between 0 and 63. The Bitrate is computed using the following formula: SWPMI_freq = SWPMI_clk / (((BitRate) + 1) * 4)

- *uint32_t SWPMI_InitTypeDef::TxBufferingMode*

Specifies the transmission buffering mode. This parameter can be a value of **SWPMI Tx Buffering Mode**

- *uint32_t SWPMI_InitTypeDef::RxBufferingMode*

Specifies the reception buffering mode. This parameter can be a value of **SWPMI Rx Buffering Mode**

72.1.2 SWPMI_HandleTypeDef

Data Fields

- *SWPMI_TypeDef * Instance*
- *SWPMI_InitTypeDef Init*
- *uint32_t * pTxBuffPtr*
- *uint32_t TxXferSize*
- *uint32_t TxXferCount*
- *uint32_t * pRxBuffPtr*
- *uint32_t RxXferSize*
- *uint32_t RxXferCount*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *_IO HAL_SWPMI_StateTypeDef State*
- *_IO uint32_t ErrorCode*

Field Documentation

- *SWPMI_TypeDef* SWPMI_HandleTypeDef::Instance*

- *SWPMI_InitTypeDef SWPMI_HandleTypeDef::Init*

- *uint32_t* SWPMI_HandleTypeDef::pTxBuffPtr*

- `uint32_t SWPMI_HandleTypeDef::TxXferSize`
- `uint32_t SWPMI_HandleTypeDef::TxXferCount`
- `uint32_t* SWPMI_HandleTypeDef::pRxBuffPtr`
- `uint32_t SWPMI_HandleTypeDef::RxXferSize`
- `uint32_t SWPMI_HandleTypeDef::RxXferCount`
- `DMA_HandleTypeDef* SWPMI_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* SWPMI_HandleTypeDef::hdmarx`
- `HAL_LockTypeDef SWPMI_HandleTypeDef::Lock`
- `__IO HAL_SWPMI_StateTypeDef SWPMI_HandleTypeDef::State`
- `__IO uint32_t SWPMI_HandleTypeDef::ErrorCode`

72.2 SWPMI Firmware driver API description

72.2.1 How to use this driver

The SWPMI HAL driver can be used as follows:

1. Declare a SWPMI_HandleTypeDef handle structure (eg. `SWPMI_HandleTypeDef hswpmi`).
2. Initialize the SWPMI low level resources by implementing the `HAL_SWPMI_MsplInit()` API:
 - a. Enable the SWPMIx interface clock with `__HAL_RCC_SWPMIx_CLK_ENABLE()`.
 - b. SWPMI IO configuration:
 - Enable the clock for the SWPMI GPIO.
 - Configure these SWPMI pins as alternate function pull-up.
 - c. NVIC configuration if you need to use interrupt process (`HAL_SWPMI_Transmit_IT()` and `HAL_SWPMI_Receive_IT()` APIs):
 - Configure the SWPMIx interrupt priority with `HAL_NVIC_SetPriority()`.
 - Enable the NVIC SWPMI IRQ handle with `HAL_NVIC_EnableIRQ()`.
 - d. DMA Configuration if you need to use DMA process (`HAL_SWPMI_Transmit_DMA()` and `HAL_SWPMI_Receive_DMA()` APIs):
 - Declare a DMA handle structure for the Tx/Rx channels.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channels and requests.
 - Associate the initialized DMA handle to the SWPMI DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channels.
3. Program the Bite Rate, Tx Buffering mode, Rx Buffering mode in the Init structure.
4. Enable the SWPMI peripheral by calling the `HAL_SWPMI_Init()` function.

72.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the SWPMI peripheral.
- De-initialize the SWPMI peripheral.

This section contains the following APIs:

- [**HAL_SWPMI_Init**](#)
- [**HAL_SWPMI_DelInit**](#)
- [**HAL_SWPMI_MspInit**](#)
- [**HAL_SWPMI_MspDelInit**](#)

72.2.3 IO operation methods

This subsection provides a set of functions allowing to manage the SWPMI data transfers.

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - Non-Blocking mode: The communication is performed using Interrupts or DMA. The end of the data processing will be indicated through the dedicated SWPMI Interrupt handler (HAL_SWPMI_IRQHandler()) when using Interrupt mode or the selected DMA channel interrupt handler when using DMA mode. The HAL_SWPMI_TxCpltCallback(), HAL_SWPMI_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or receive process. The HAL_SWPMI_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode API's are:
 - HAL_SWPMI_Transmit()
 - HAL_SWPMI_Receive()
3. Non-Blocking mode API's with Interrupt are:
 - HAL_SWPMI_Transmit_IT()
 - HAL_SWPMI_Receive_IT()
 - HAL_SWPMI_IRQHandler()
4. Non-Blocking mode API's with DMA are:
 - HAL_SWPMI_Transmit_DMA()
 - HAL_SWPMI_Receive_DMA()
 - HAL_SWPMI_DMAPause()
 - HAL_SWPMI_DMAResume()
 - HAL_SWPMI_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non-Blocking mode:
 - HAL_SWPMI_TxHalfCpltCallback()
 - HAL_SWPMI_TxCpltCallback()
 - HAL_SWPMI_RxHalfCpltCallback()
 - HAL_SWPMI_RxCpltCallback()
 - HAL_SWPMI_ErrorCallback()
6. The capability to launch the above IO operations in loopback mode for user application verification:
 - HAL_SWPMI_EnableLoopback()
 - HAL_SWPMI_DisableLoopback()

This section contains the following APIs:

- [**HAL_SWPMI_Transmit**](#)
- [**HAL_SWPMI_Receive**](#)
- [**HAL_SWPMI_Transmit_IT**](#)
- [**HAL_SWPMI_Receive_IT**](#)
- [**HAL_SWPMI_Transmit_DMA**](#)
- [**HAL_SWPMI_Receive_DMA**](#)
- [**HAL_SWPMI_DMAStop**](#)
- [**HAL_SWPMI_EnableLoopback**](#)
- [**HAL_SWPMI_DisableLoopback**](#)

72.2.4 SWPMI IRQ handler and callbacks

This section provides SWPMI IRQ handler and callback functions called within the IRQ handler.

This section contains the following APIs:

- [**HAL_SWPMI_IRQHandler**](#)
- [**HAL_SWPMI_TxCpltCallback**](#)
- [**HAL_SWPMI_TxHalfCpltCallback**](#)
- [**HAL_SWPMI_RxCpltCallback**](#)
- [**HAL_SWPMI_RxHalfCpltCallback**](#)
- [**HAL_SWPMI_ErrorCallback**](#)

72.2.5 Peripheral Control methods

This subsection provides a set of functions allowing to control the SWPMI.

- [**HAL_SWPMI_GetState\(\)**](#) API is helpful to check in run-time the state of the SWPMI peripheral
- [**HAL_SWPMI_GetError\(\)**](#) API is helpful to check in run-time the error state of the SWPMI peripheral

This section contains the following APIs:

- [**HAL_SWPMI_GetState**](#)
- [**HAL_SWPMI_GetError**](#)

72.2.6 Detailed description of functions

HAL_SWPMI_Init

Function name

HAL_StatusTypeDef HAL_SWPMI_Init (SWPMI_HandleTypeDef * hswpmi)

Function description

Initialize the SWPMI peripheral according to the specified parameters in the **SWPMI_InitTypeDef**.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL:** status

HAL_SWPMI_DeInit

Function name

HAL_StatusTypeDef HAL_SWPMI_DeInit (SWPMI_HandleTypeDef * hswpmi)

Function description

De-initialize the SWPMI peripheral.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL:** status

HAL_SWPMI_MspInit

Function name

void HAL_SWPMI_MspInit (SWPMI_HandleTypeDef * hswpmi)

Function description

Initialize the SWPMI MSP.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_MspDeInit

Function name

void HAL_SWPMI_MspDeInit (SWPMI_HandleTypeDef * hswpmi)

Function description

Deinitialize the SWPMI MSP.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_Transmit

Function name

HAL_StatusTypeDef HAL_SWPMI_Transmit (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hswpmi:** pointer to a SWPMI_HandleTypeDef structure that contains the configuration information for SWPMI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SWPMI_Receive

Function name

HAL_StatusTypeDef HAL_SWPMI_Receive (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hswpmi:** pointer to a SWPMI_HandleTypeDef structure that contains the configuration information for SWPMI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SWPMI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SWPMI_Transmit_IT (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with interrupt.

Parameters

- **hswpmi:** pointer to a SWPMI_HandleTypeDef structure that contains the configuration information for SWPMI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SWPMI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SWPMI_Receive_IT (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hswpmi:** SWPMI handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be received

Return values

- **HAL:** status

HAL_SWPMI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SWPMI_Transmit_DMA (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA interrupt.

Parameters

- **hswpmi:** SWPMI handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- **HAL:** status

HAL_SWPMI_Receive_DMA

Function name

```
HAL_StatusTypeDef HAL_SWPMI_Receive_DMA (SWPMI_HandleTypeDef * hswpmi, uint32_t * pData,  
uint16_t Size)
```

Function description

Receive an amount of data in non-blocking mode with DMA interrupt.

Parameters

- **hswpmi:** SWPMI handle
- **pData:** pointer to data buffer
- **Size:** amount of data to be received

Return values

- **HAL:** status

HAL_SWPMI_DMASStop

Function name

```
HAL_StatusTypeDef HAL_SWPMI_DMASStop (SWPMI_HandleTypeDef * hswpmi)
```

Function description

Stop all DMA transfers.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL_OK:**

HAL_SWPMI_EnableLoopback

Function name

```
HAL_StatusTypeDef HAL_SWPMI_EnableLoopback (SWPMI_HandleTypeDef * hswpmi)
```

Function description

Enable the Loopback mode.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL_OK:** / **HAL_BUSY**

Notes

- Loopback mode is to be used only for test purposes

HAL_SWPMI_DisableLoopback

Function name

```
HAL_StatusTypeDef HAL_SWPMI_DisableLoopback (SWPMI_HandleTypeDef * hswpmi)
```

Function description

Disable the Loopback mode.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL_OK:** / **HAL_BUSY**

Notes

- Loopback mode is to be used only for test purposes

HAL_SWPMI_IRQHandler**Function name**

```
void HAL_SWPMI_IRQHandler (SWPMI_HandleTypeDef * hswpmi)
```

Function description

Handle SWPMI interrupt request.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_TxCpltCallback**Function name**

```
void HAL_SWPMI_TxCpltCallback (SWPMI_HandleTypeDef * hswpmi)
```

Function description

Tx Transfer completed callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_TxHalfCpltCallback**Function name**

```
void HAL_SWPMI_TxHalfCpltCallback (SWPMI_HandleTypeDef * hswpmi)
```

Function description

Tx Half Transfer completed callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_RxCpltCallback**Function name**

```
void HAL_SWPMI_RxCpltCallback (SWPMI_HandleTypeDef * hswpmi)
```

Function description

Rx Transfer completed callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_RxHalfCpltCallback

Function name

void HAL_SWPMI_RxHalfCpltCallback (SWPMI_HandleTypeDef * hswpmi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_ErrorCallback

Function name

void HAL_SWPMI_ErrorCallback (SWPMI_HandleTypeDef * hswpmi)

Function description

SWPMI error callback.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **None:**

HAL_SWPMI_GetState

Function name

HAL_SWPMI_StateTypeDef HAL_SWPMI_GetState (SWPMI_HandleTypeDef * hswpmi)

Function description

Return the SWPMI handle state.

Parameters

- **hswpmi:** SWPMI handle

Return values

- **HAL:** state

HAL_SWPMI_GetError

Function name

uint32_t HAL_SWPMI_GetError (SWPMI_HandleTypeDef * hswpmi)

Function description

Return the SWPMI error code.

Parameters

- **hswpmi:** : pointer to a SWPMI_HandleTypeDef structure that contains the configuration information for the specified SWPMI.

Return values

- **SWPMI:** Error Code

72.3 SWPMI Firmware driver defines

72.3.1 SWPMI

SWPMI Error Code Bitmap

HAL_SWPMI_ERROR_NONE

No error

HAL_SWPMI_ERROR_CRC

frame error

HAL_SWPMI_ERROR_OVR

Overrun error

HAL_SWPMI_ERROR_UDR

Underrun error

HAL_SWPMI_ERROR_DMA

DMA transfer error

SWPMI Exported Macros

_HAL_SWPMI_RESET_HANDLE_STATE

Description:

- Reset SWPMI handle state.

Parameters:

- **_HANDLE_**: specifies the SWPMI Handle.

Return value:

- None

_HAL_SWPMI_ENABLE

Description:

- Enable the SWPMI peripheral.

Parameters:

- **_HANDLE_**: SWPMI handle

Return value:

- None

_HAL_SWPMI_DISABLE

Description:

- Disable the SWPMI peripheral.

Parameters:

- `__HANDLE__`: SWPMI handle

Return value:

- None

[__HAL_SWPMI_TRANSCEIVER_ENABLE](#)**Description:**

- Enable/Disable the SWPMI transceiver.

Parameters:

- `__HANDLE__`: SWPMI handle

Return value:

- None

[__HAL_SWPMI_TRANSCEIVER_DISABLE](#)[__HAL_SWPMI_GET_FLAG](#)**Description:**

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

Parameters:

- `__HANDLE__`: specifies the SWPMI Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `SWPMI_FLAG_RXBFF` : Receive buffer full flag.
 - `SWPMI_FLAG_TXBEF` : Transmit buffer empty flag.
 - `SWPMI_FLAG_RXBERF` : Receive CRC error flag.
 - `SWPMI_FLAG_RXOVRF` : Receive overrun error flag.
 - `SWPMI_FLAG_TXUNRF` : Transmit underrun error flag.
 - `SWPMI_FLAG_RXNE` : Receive data register not empty.
 - `SWPMI_FLAG_TXE` : Transmit data register empty.
 - `SWPMI_FLAG_TCF` : Transfer complete flag.
 - `SWPMI_FLAG_SRF` : Slave resume flag.
 - `SWPMI_FLAG_SUSP` : SUSPEND flag.
 - `SWPMI_FLAG_DEACTF` : DEACTIVATED flag.

Return value:

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

[__HAL_SWPMI_CLEAR_FLAG](#)**Description:**

- Clear the specified SWPMI ISR flag.

Parameters:

- `__HANDLE__`: specifies the SWPMI Handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be one of the following values:
 - `SWPMI_FLAG_RXBFF` : Receive buffer full flag.
 - `SWPMI_FLAG_TXBEF` : Transmit buffer empty flag.
 - `SWPMI_FLAG_RXBERF` : Receive CRC error flag.
 - `SWPMI_FLAG_RXOVRF` : Receive overrun error flag.
 - `SWPMI_FLAG_TXUNRF` : Transmit underrun error flag.

- SWPMI_FLAG_TCF : Transfer complete flag.
- SWPMI_FLAG_SRF : Slave resume flag.
- SWPMI_FLAG_RDYF: Transceiver ready flag

Return value:

- None

[__HAL_SWPMI_ENABLE_IT](#)**Description:**

- Enable the specified SWPMI interrupt.

Parameters:

- [__HANDLE__](#): specifies the SWPMI Handle.
- [__INTERRUPT__](#): specifies the SWPMI interrupt source to enable. This parameter can be one of the following values:
 - SWPMI_IT_SRIE : Slave resume interrupt.
 - SWPMI_IT_TCIE : Transmit complete interrupt.
 - SWPMI_IT_TIE : Transmit interrupt.
 - SWPMI_IT_RIE : Receive interrupt.
 - SWPMI_IT_TXUNRIE : Transmit underrun error interrupt.
 - SWPMI_IT_RXOVRIE : Receive overrun error interrupt.
 - SWPMI_IT_RXBEIE : Receive CRC error interrupt.
 - SWPMI_IT_TXBEIE : Transmit buffer empty interrupt.
 - SWPMI_IT_RXBFIE : Receive buffer full interrupt.

Return value:

- None

[__HAL_SWPMI_DISABLE_IT](#)**Description:**

- Disable the specified SWPMI interrupt.

Parameters:

- [__HANDLE__](#): specifies the SWPMI Handle.
- [__INTERRUPT__](#): specifies the SWPMI interrupt source to disable. This parameter can be one of the following values:
 - SWPMI_IT_SRIE : Slave resume interrupt.
 - SWPMI_IT_TCIE : Transmit complete interrupt.
 - SWPMI_IT_TIE : Transmit interrupt.
 - SWPMI_IT_RIE : Receive interrupt.
 - SWPMI_IT_TXUNRIE : Transmit underrun error interrupt.
 - SWPMI_IT_RXOVRIE : Receive overrun error interrupt.
 - SWPMI_IT_RXBEIE : Receive CRC error interrupt.
 - SWPMI_IT_TXBEIE : Transmit buffer empty interrupt.
 - SWPMI_IT_RXBFIE : Receive buffer full interrupt.

Return value:

- None

[__HAL_SWPMI_GET_IT](#)**Description:**

- Check whether the specified SWPMI interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the SWPMI Handle.
- `__IT__`: specifies the SWPMI interrupt to check. This parameter can be one of the following values:
 - `SWPMI_IT_SRIE` : Slave resume interrupt.
 - `SWPMI_IT_TCIE` : Transmit complete interrupt.
 - `SWPMI_IT_TIE` : Transmit interrupt.
 - `SWPMI_IT_RIE` : Receive interrupt.
 - `SWPMI_IT_TXUNRIE` : Transmit underrun error interrupt.
 - `SWPMI_IT_RXOVRIE` : Receive overrun error interrupt.
 - `SWPMI_IT_RXBERIE` : Receive CRC error interrupt.
 - `SWPMI_IT_TXBEIE` : Transmit buffer empty interrupt.
 - `SWPMI_IT_RXBFIE` : Receive buffer full interrupt.

Return value:

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

[`__HAL_SWPMI_GET_IT_SOURCE`](#)**Description:**

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

Parameters:

- `__HANDLE__`: specifies the SWPMI Handle.
- `__IT__`: specifies the SWPMI interrupt source to check. This parameter can be one of the following values:
 - `SWPMI_IT_SRIE` : Slave resume interrupt.
 - `SWPMI_IT_TCIE` : Transmit complete interrupt.
 - `SWPMI_IT_TIE` : Transmit interrupt.
 - `SWPMI_IT_RIE` : Receive interrupt.
 - `SWPMI_IT_TXUNRIE` : Transmit underrun error interrupt.
 - `SWPMI_IT_RXOVRIE` : Receive overrun error interrupt.
 - `SWPMI_IT_RXBERIE` : Receive CRC error interrupt.
 - `SWPMI_IT_TXBEIE` : Transmit buffer empty interrupt.
 - `SWPMI_IT_RXBFIE` : Receive buffer full interrupt.

Return value:

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

SWPMI Status Flags[`SWPMI_FLAG_RXBFF`](#)[`SWPMI_FLAG_TXBEF`](#)[`SWPMI_FLAG_RXBERF`](#)[`SWPMI_FLAG_RXOVRF`](#)[`SWPMI_FLAG_TXUNRF`](#)[`SWPMI_FLAG_RXNE`](#)[`SWPMI_FLAG_TXE`](#)[`SWPMI_FLAG_TCF`](#)

`SWPMI_FLAG_SRF`

`SWPMI_FLAG_SUSP`

`SWPMI_FLAG_DEACTF`

`SWPMI_FLAG_RDYF`

SWPMI Interrupts Definition

`SWPMI_IT_RDYIE`

`SWPMI_IT_SRIE`

`SWPMI_IT_TCIE`

`SWPMI_IT_TIE`

`SWPMI_IT_RIE`

`SWPMI_IT_TXUNRIE`

`SWPMI_IT_RXOVRIE`

`SWPMI_IT_RXBERIE`

`SWPMI_IT_TXBEIE`

`SWPMI_IT_RXBFIE`

SWPMI Rx Buffering Mode

`SWPMI_RX_NO_SOFTWAREBUFFER`

`SWPMI_RX_SINGLE_SOFTWAREBUFFER`

`SWPMI_RX_MULTI_SOFTWAREBUFFER`

SWPMI Tx Buffering Mode

`SWPMI_TX_NO_SOFTWAREBUFFER`

`SWPMI_TX_SINGLE_SOFTWAREBUFFER`

`SWPMI_TX_MULTI_SOFTWAREBUFFER`

SWPMI Voltage Class

`SWPMI_VOLTAGE_CLASS_C`

`SWPMI_VOLTAGE_CLASS_B`

73 HAL TIM Generic Driver

73.1 TIM Firmware driver registers structures

73.1.1 TIM_Base_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t CounterMode*
- *uint32_t Period*
- *uint32_t ClockDivision*
- *uint32_t RepetitionCounter*
- *uint32_t AutoReloadPreload*

Field Documentation

- *uint32_t TIM_Base_InitTypeDef::Prescaler*

Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

- *uint32_t TIM_Base_InitTypeDef::CounterMode*

Specifies the counter mode. This parameter can be a value of **TIM Counter Mode**

- *uint32_t TIM_Base_InitTypeDef::Period*

Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

- *uint32_t TIM_Base_InitTypeDef::ClockDivision*

Specifies the clock division. This parameter can be a value of **TIM Clock Division**

- *uint32_t TIM_Base_InitTypeDef::RepetitionCounter*

Specifies the repetition counter value. Each time the RCR down-counter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:

- the number of PWM periods in edge-aligned mode
- the number of half PWM period in center-aligned mode This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

Note:

- This parameter is valid only for TIM1 and TIM8.

- *uint32_t TIM_Base_InitTypeDef::AutoReloadPreload*

Specifies the auto-reload preload. This parameter can be a value of **TIM Auto-Reload Preload**

73.1.2 TIM_OC_InitTypeDef

Data Fields

- *uint32_t OCMode*
- *uint32_t Pulse*
- *uint32_t OCPolarity*
- *uint32_t OCNPolarity*
- *uint32_t OCFastMode*

- `uint32_t OCIdleState`
- `uint32_t OCNIdleState`

Field Documentation

- `uint32_t TIM_OC_InitTypeDef::OCMode`

Specifies the TIM mode. This parameter can be a value of **TIM Output Compare and PWM Modes**

- `uint32_t TIM_OC_InitTypeDef::Pulse`

Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

- `uint32_t TIM_OC_InitTypeDef::OCPolarity`

Specifies the output polarity. This parameter can be a value of **TIM Output Compare Polarity**

- `uint32_t TIM_OC_InitTypeDef::OCNPolarity`

Specifies the complementary output polarity. This parameter can be a value of **TIM Complementary Output Compare Polarity**

Note:

– This parameter is valid only for TIM1 and TIM8.

- `uint32_t TIM_OC_InitTypeDef::OCFastMode`

Specifies the Fast mode state. This parameter can be a value of **TIM Output Fast State**

Note:

– This parameter is valid only in PWM1 and PWM2 mode.

- `uint32_t TIM_OC_InitTypeDef::OCIdleState`

Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of **TIM Output Compare Idle State**

Note:

– This parameter is valid only for TIM1 and TIM8.

- `uint32_t TIM_OC_InitTypeDef::OCNIdleState`

Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of **TIM Complementary Output Compare Idle State**

Note:

– This parameter is valid only for TIM1 and TIM8.

73.1.3 TIM_OnePulse_InitTypeDef

Data Fields

- `uint32_t OCMode`
- `uint32_t Pulse`
- `uint32_t OCPolarity`
- `uint32_t OCNPolarity`
- `uint32_t OCIdleState`
- `uint32_t OCNIdleState`
- `uint32_t IC_Polarity`
- `uint32_t IC_Selection`
- `uint32_t IC_Filter`

Field Documentation

- `uint32_t TIM_OnePulse_InitTypeDef::OCMode`

Specifies the TIM mode. This parameter can be a value of **TIM Output Compare and PWM Modes**

- **`uint32_t TIM_OnePulse_InitTypeDef::Pulse`**

Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

- **`uint32_t TIM_OnePulse_InitTypeDef::OCPolarity`**

Specifies the output polarity. This parameter can be a value of **TIM Output Compare Polarity**

- **`uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity`**

Specifies the complementary output polarity. This parameter can be a value of **TIM Complementary Output Compare Polarity**

Note:

— This parameter is valid only for TIM1 and TIM8.

- **`uint32_t TIM_OnePulse_InitTypeDef::OCIdleState`**

Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of **TIM Output Compare Idle State**

Note:

— This parameter is valid only for TIM1 and TIM8.

- **`uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState`**

Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of **TIM Complementary Output Compare Idle State**

Note:

— This parameter is valid only for TIM1 and TIM8.

- **`uint32_t TIM_OnePulse_InitTypeDef::ICPolarity`**

Specifies the active edge of the input signal. This parameter can be a value of **TIM Input Capture Polarity**

- **`uint32_t TIM_OnePulse_InitTypeDef::ICSelection`**

Specifies the input. This parameter can be a value of **TIM Input Capture Selection**

- **`uint32_t TIM_OnePulse_InitTypeDef::ICFilter`**

Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

73.1.4 TIM_IC_InitTypeDef

Data Fields

- **`uint32_t ICPolarity`**
- **`uint32_t ICSelection`**
- **`uint32_t ICPrescaler`**
- **`uint32_t ICFilter`**

Field Documentation

- **`uint32_t TIM_IC_InitTypeDef::ICPolarity`**

Specifies the active edge of the input signal. This parameter can be a value of **TIM Input Capture Polarity**

- **`uint32_t TIM_IC_InitTypeDef::ICSelection`**

Specifies the input. This parameter can be a value of **TIM Input Capture Selection**

- **`uint32_t TIM_IC_InitTypeDef::ICPrescaler`**

Specifies the Input Capture Prescaler. This parameter can be a value of **TIM Input Capture Prescaler**

- **`uint32_t TIM_IC_InitTypeDef::ICFilter`**

Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

73.1.5 TIM_Encoder_InitTypeDef

Data Fields

- `uint32_t EncoderMode`
- `uint32_t IC1Polarity`
- `uint32_t IC1Selection`
- `uint32_t IC1Prescaler`
- `uint32_t IC1Filter`
- `uint32_t IC2Polarity`
- `uint32_t IC2Selection`
- `uint32_t IC2Prescaler`
- `uint32_t IC2Filter`

Field Documentation

- `uint32_t TIM_Encoder_InitTypeDef::EncoderMode`

Specifies the active edge of the input signal. This parameter can be a value of **TIM Encoder Mode**

- `uint32_t TIM_Encoder_InitTypeDef::IC1Polarity`

Specifies the active edge of the input signal. This parameter can be a value of **TIM Input Capture Polarity**

- `uint32_t TIM_Encoder_InitTypeDef::IC1Selection`

Specifies the input. This parameter can be a value of **TIM Input Capture Selection**

- `uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler`

Specifies the Input Capture Prescaler. This parameter can be a value of **TIM Input Capture Prescaler**

- `uint32_t TIM_Encoder_InitTypeDef::IC1Filter`

Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

- `uint32_t TIM_Encoder_InitTypeDef::IC2Polarity`

Specifies the active edge of the input signal. This parameter can be a value of **TIM Input Capture Polarity**

- `uint32_t TIM_Encoder_InitTypeDef::IC2Selection`

Specifies the input. This parameter can be a value of **TIM Input Capture Selection**

- `uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler`

Specifies the Input Capture Prescaler. This parameter can be a value of **TIM Input Capture Prescaler**

- `uint32_t TIM_Encoder_InitTypeDef::IC2Filter`

Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

73.1.6 TIM_ClockConfigTypeDef

Data Fields

- `uint32_t ClockSource`
- `uint32_t ClockPolarity`
- `uint32_t ClockPrescaler`
- `uint32_t ClockFilter`

Field Documentation

- `uint32_t TIM_ClockConfigTypeDef::ClockSource`

TIM clock sources This parameter can be a value of ***TIM Clock Source***

- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***

TIM clock polarity This parameter can be a value of ***TIM Clock Polarity***

- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***

TIM clock prescaler This parameter can be a value of ***TIM Clock Prescaler***

- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***

TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

73.1.7 **TIM_ClearInputConfigTypeDef**

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***

TIM clear Input state This parameter can be ENABLE or DISABLE

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***

TIM clear Input sources This parameter can be a value of ***TIM Extended Clear Input Source***

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***

TIM Clear Input polarity This parameter can be a value of ***TIM Clear Input Polarity***

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***

TIM Clear Input prescaler This parameter can be a value of ***TIM Clear Input Prescaler***

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***

TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

73.1.8 **TIM_MasterConfigTypeDef**

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterOutputTrigger2***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***

Trigger output (TRGO) selection This parameter can be a value of ***TIM Master Mode Selection***

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger2***

Trigger output2 (TRGO2) selection This parameter can be a value of ***TIM Master Mode Selection 2 (TRGO2)***

- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***

Master/slave mode selection This parameter can be a value of ***TIM Master/Slave Mode***

73.1.9 **TIM_SlaveConfigTypeDef**

Data Fields

- `uint32_t SlaveMode`
- `uint32_t InputTrigger`
- `uint32_t TriggerPolarity`
- `uint32_t TriggerPrescaler`
- `uint32_t TriggerFilter`

Field Documentation

- `uint32_t TIM_SlaveConfigTypeDef::SlaveMode`
Slave mode selection This parameter can be a value of **TIM Slave mode**
- `uint32_t TIM_SlaveConfigTypeDef::InputTrigger`
Input Trigger source This parameter can be a value of **TIM Trigger Selection**
- `uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity`
Input Trigger polarity This parameter can be a value of **TIM Trigger Polarity**
- `uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler`
Input trigger prescaler This parameter can be a value of **TIM Trigger Prescaler**
- `uint32_t TIM_SlaveConfigTypeDef::TriggerFilter`
Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

73.1.10 TIM_BreakDeadTimeConfigTypeDef

Data Fields

- `uint32_t OffStateRunMode`
- `uint32_t OffStateIDLEMode`
- `uint32_t LockLevel`
- `uint32_t DeadTime`
- `uint32_t BreakState`
- `uint32_t BreakPolarity`
- `uint32_t BreakFilter`
- `uint32_t Break2State`
- `uint32_t Break2Polarity`
- `uint32_t Break2Filter`
- `uint32_t AutomaticOutput`

Field Documentation

- `uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode`
TIM off state in run mode This parameter can be a value of **TIM Off-state Selection for Run Mode**
- `uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode`
TIM off state in IDLE mode This parameter can be a value of **TIM Off-state Selection for Idle Mode**
- `uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel`
TIM Lock level This parameter can be a value of **TIM Lock Configuration**
- `uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime`
TIM dead Time This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- `uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState`
TIM Break State This parameter can be a value of **TIM Break Input Enable**
- `uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity`
TIM Break input polarity This parameter can be a value of **TIM Break Input Polarity**

- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakFilter`**
Specifies the break input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2State`**
TIM Break2 State This parameter can be a value of **`TIMEX Break input 2 Enable`**
- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Polarity`**
TIM Break2 input polarity This parameter can be a value of **`TIM Extended Break Input 2 Polarity`**
- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Filter`**
TIM break2 input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- **`uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput`**
TIM Automatic Output Enable state This parameter can be a value of **`TIM Automatic Output Enable`**

73.1.11 **TIM_HandleTypeDef**

Data Fields

- **`TIM_TypeDef * Instance`**
- **`TIM_Base_InitTypeDef Init`**
- **`HAL_TIM_ActiveChannel Channel`**
- **`DMA_HandleTypeDef * hdma`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_TIM_StateTypeDef State`**

Field Documentation

- **`TIM_TypeDef* TIM_HandleTypeDef::Instance`**
Register base address
- **`TIM_Base_InitTypeDef TIM_HandleTypeDef::Init`**
TIM Time Base required parameters
- **`HAL_TIM_ActiveChannel TIM_HandleTypeDef::Channel`**
Active channel
- **`DMA_HandleTypeDef* TIM_HandleTypeDef::hdma[7]`**
DMA Handlers array This array is accessed by a **`DMA_HandleTypeDef`**
- **`HAL_LockTypeDef TIM_HandleTypeDef::Lock`**
Locking object
- **`__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State`**
TIM operation state

73.2 TIM Firmware driver API description

73.2.1 **TIM Generic features**

The Timer features include:

1. 16-bit (32-bit for TIM2/TIM5) up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:

- Input Capture
- Output Compare
- PWM generation (Edge and Center-aligned Mode)
- One-pulse mode output

73.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
 - Time Base : HAL_TIM_Base_MspInit()
 - Input Capture : HAL_TIM_IC_MspInit()
 - Output Compare : HAL_TIM_OC_MspInit()
 - PWM generation : HAL_TIM_PWM_MspInit()
 - One-pulse mode output : HAL_TIM_OnePulse_MspInit()
 - Encoder mode output : HAL_TIM_Encoder_MspInit()
2. Initialize the TIM low level resources :
 - a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
 - b. TIM pins configuration
 - Enable the clock for the TIM GPIOs using the following function:
__HAL_RCC_GPIOx_CLK_ENABLE();
 - Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
 - HAL_TIM_Base_Init: to use the Timer to generate a simple time base.
 - HAL_TIM_OC_Init and HAL_TIM_OC_ConfigChannel: to use the Timer to generate an Output Compare signal.
 - HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
 - HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
 - HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
 - HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
 - Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT().
 - Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT().
 - Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT().
 - PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT().
 - One-pulse mode output : HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT().
 - Encoder mode output : HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions:
 - HAL_TIM_DMABurst_WriteStart().
 - HAL_TIM_DMABurst_ReadStart().

73.2.3 TIM Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the TIM Output Compare.
- Stop the TIM Output Compare.

- Start the TIM Output Compare and enable interrupt.
- Stop the TIM Output Compare and disable interrupt.
- Start the TIM Output Compare and enable DMA transfer.
- Stop the TIM Output Compare and disable DMA transfer.

This section contains the following APIs:

- [HAL_TIM_OC_Init](#)
- [HAL_TIM_OC_DelInit](#)
- [HAL_TIM_OC_MspInit](#)
- [HAL_TIM_OC_MspDelInit](#)
- [HAL_TIM_OC_Start](#)
- [HAL_TIM_OC_Stop](#)
- [HAL_TIM_OC_Start_IT](#)
- [HAL_TIM_OC_Stop_IT](#)
- [HAL_TIM_OC_Start_DMA](#)
- [HAL_TIM_OC_Stop_DMA](#)

73.2.4

TIM PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM mode.
- De-initialize the TIM PWM mode.
- Start the TIM PWM mode.
- Stop the TIM PWM mode.
- Start the TIM PWM mode and enable interrupt.
- Stop the TIM PWM mode and disable interrupt.
- Start the TIM PWM mode and enable DMA transfer.
- Stop the TIM PWM mode and disable DMA transfer.

This section contains the following APIs:

- [HAL_TIM_PWM_Init](#)
- [HAL_TIM_PWM_DelInit](#)
- [HAL_TIM_PWM_MspInit](#)
- [HAL_TIM_PWM_MspDelInit](#)
- [HAL_TIM_PWM_Start](#)
- [HAL_TIM_PWM_Stop](#)
- [HAL_TIM_PWM_Start_IT](#)
- [HAL_TIM_PWM_Stop_IT](#)
- [HAL_TIM_PWM_Start_DMA](#)
- [HAL_TIM_PWM_Stop_DMA](#)

73.2.5

TIM Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the TIM Input Capture mode.
- Stop the TIM Input Capture mode.
- Start the TIM Input Capture mode and enable interrupt.
- Stop the TIM Input Capture mode and disable interrupt.
- Start the TIM Input Capture mode and enable DMA transfer.

- Stop the TIM Input Capture mode and disable DMA transfer.

This section contains the following APIs:

- [HAL_TIM_IC_Init](#)
- [HAL_TIM_IC_DeInit](#)
- [HAL_TIM_IC_MspInit](#)
- [HAL_TIM_IC_MspDeInit](#)
- [HAL_TIM_IC_Start](#)
- [HAL_TIM_IC_Stop](#)
- [HAL_TIM_IC_Start_IT](#)
- [HAL_TIM_IC_Stop_IT](#)
- [HAL_TIM_IC_Start_DMA](#)
- [HAL_TIM_IC_Stop_DMA](#)

73.2.6 TIM One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse mode.
- De-initialize the TIM One Pulse mode.
- Start the TIM One Pulse mode.
- Stop the TIM One Pulse mode.
- Start the TIM One Pulse mode and enable interrupt.
- Stop the TIM One Pulse mode and disable interrupt.
- Start the TIM One Pulse mode and enable DMA transfer.
- Stop the TIM One Pulse mode and disable DMA transfer.

This section contains the following APIs:

- [HAL_TIM_OnePulse_Init](#)
- [HAL_TIM_OnePulse_DeInit](#)
- [HAL_TIM_OnePulse_MspInit](#)
- [HAL_TIM_OnePulse_MspDeInit](#)
- [HAL_TIM_OnePulse_Start](#)
- [HAL_TIM_OnePulse_Stop](#)
- [HAL_TIM_OnePulse_Start_IT](#)
- [HAL_TIM_OnePulse_Stop_IT](#)

73.2.7 IRQ handler management

This section provides TIM IRQ handler function.

This section contains the following APIs:

- [HAL_TIM_IRQHandler](#)

73.2.8 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input/Output channels for Output Compare, PWM, Input Capture or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.

This section contains the following APIs:

- [*HAL_TIM_OC_ConfigChannel*](#)
- [*HAL_TIM_IC_ConfigChannel*](#)
- [*HAL_TIM_PWM_ConfigChannel*](#)
- [*HAL_TIM_OnePulse_ConfigChannel*](#)
- [*HAL_TIM_DMABurst_WriteStart*](#)
- [*HAL_TIM_DMABurst_WriteStop*](#)
- [*HAL_TIM_DMABurst_ReadStart*](#)
- [*HAL_TIM_DMABurst_ReadStop*](#)
- [*HAL_TIM_GenerateEvent*](#)
- [*HAL_TIM_ConfigOCrefClear*](#)
- [*HAL_TIM_ConfigClockSource*](#)
- [*HAL_TIM_ConfigT11Input*](#)
- [*HAL_TIM_SlaveConfigSynchronization*](#)
- [*HAL_TIM_SlaveConfigSynchronization_IT*](#)
- [*HAL_TIM_ReadCapturedValue*](#)

73.2.9 TIM Callbacks functions

This section provides TIM callback functions:

- TIM Period elapsed callback
- TIM Output Compare callback
- TIM Input capture callback
- TIM Trigger callback
- TIM Error callback

This section contains the following APIs:

- [*HAL_TIM_PeriodElapsedCallback*](#)
- [*HAL_TIM_OC_DelayElapsedCallback*](#)
- [*HAL_TIM_IC_CaptureCallback*](#)
- [*HAL_TIM_PWM_PulseFinishedCallback*](#)
- [*HAL_TIM_TriggerCallback*](#)
- [*HAL_TIM_ErrorCallback*](#)

73.2.10 Peripheral State functions

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

This section contains the following APIs:

- [*HAL_TIM_Base_GetState*](#)
- [*HAL_TIM_OC_GetState*](#)
- [*HAL_TIM_PWM_GetState*](#)
- [*HAL_TIM_IC_GetState*](#)
- [*HAL_TIM_OnePulse_GetState*](#)
- [*HAL_TIM_Encoder_GetState*](#)

73.2.11 Detailed description of functions

`HAL_TIM_Base_Init`

Function name

`HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)`

Function description

Initialize the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_Base_DeInit

Function name

```
HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)
```

Function description

Deinitialize the TIM Base peripheral.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_Base_MspInit

Function name

```
void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)
```

Function description

Initialize the TIM Time Base MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_Base_MspDeInit

Function name

```
void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)
```

Function description

Deinitialize TIM Time Base MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_Base_Start

Function name

```
HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)
```

Function description

Starts the TIM Time Base generation.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_Base_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Time Base generation.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_Base_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Time Base generation in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_Base_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Time Base generation in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_Base_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Time Base generation in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to peripheral.

Return values

- **HAL:** status

HAL_TIM_Base_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Time Base generation in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_OC_Init

Function name

HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)

Function description

Initialize the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_OC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitialize the TIM peripheral.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_OC_MspInit

Function name

```
void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)
```

Function description

Initialize the TIM Output Compare MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_OC_MspDeInit

Function name

```
void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)
```

Function description

DeInitialize TIM Output Compare MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_OC_Start

Function name

```
HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Starts the TIM Output Compare signal generation.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop

Function name

```
HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Stops the TIM Output Compare signal generation.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_OC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Init

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)

Function description

Initialize the TIM PWM mode according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_PWM_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitialize the TIM peripheral.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_PWM_MspInit

Function name

```
void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)
```

Function description

Initialize the TIM PWM MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_PWM_MspDeInit

Function name

```
void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)
```

Function description

DeInitialize TIM PWM MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_PWM_Start

Function name

```
HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Starts the PWM signal generation.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - **TIM_CHANNEL_1:** TIM Channel 1 selected
 - **TIM_CHANNEL_2:** TIM Channel 2 selected
 - **TIM_CHANNEL_3:** TIM Channel 3 selected
 - **TIM_CHANNEL_4:** TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop

Function name

`HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)`

Function description

Stops the PWM signal generation.

Parameters

- **htim:** pointer to a `TIM_HandleTypeDef` structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected
 - `TIM_CHANNEL_3`: TIM Channel 3 selected
 - `TIM_CHANNEL_4`: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_IT

Function name

`HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)`

Function description

Starts the PWM signal generation in interrupt mode.

Parameters

- **htim:** pointer to a `TIM_HandleTypeDef` structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected
 - `TIM_CHANNEL_3`: TIM Channel 3 selected
 - `TIM_CHANNEL_4`: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_IT

Function name

`HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)`

Function description

Stops the PWM signal generation in interrupt mode.

Parameters

- **htim:** pointer to a `TIM_HandleTypeDef` structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected
 - `TIM_CHANNEL_3`: TIM Channel 3 selected
 - `TIM_CHANNEL_4`: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Init

Function name

HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)

Function description

Initialize the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_IC_DeInit**Function name**

```
HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)
```

Function description

DeInitialize the TIM peripheral.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_IC_MspInit**Function name**

```
void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)
```

Function description

Initialize the TIM Input Capture MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_IC_MspDeInit**Function name**

```
void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)
```

Function description

DeInitialize TIM Input Capture MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_IC_Start**Function name**

```
HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Starts the TIM Input Capture measurement.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - **TIM_CHANNEL_1:** TIM Channel 1 selected

- TIM_CHANNEL_2: TIM Channel 2 selected
- TIM_CHANNEL_3: TIM Channel 3 selected
- TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

- **Channel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_DMA

Function name

```
HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t *  
pData, uint16_t Length)
```

Function description

Starts the TIM Input Capture measurement on in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_IC_Stop_DMA

Function name

```
HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Stops the TIM Input Capture measurement on in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Init

Function name

```
HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)
```

Function description

Initialize the TIM One Pulse mode according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values:
 - TIM_OPMODE_SINGLE: Only one pulse will be generated.
 - TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.

Return values

- **HAL:** status

HAL_TIM_OnePulse_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitialize the TIM One Pulse mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_OnePulse_MspInit

Function name

void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)

Function description

Initialize the TIM One Pulse MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_OnePulse_MspDeInit

Function name

void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitialize TIM One Pulse MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_OnePulse_Start

Function name

`HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)`

Function description

Starts the TIM One Pulse signal generation.

Parameters

- **htim:** pointer to a `TIM_HandleTypeDef` structure that contains the configuration information for TIM module.
- **OutputChannel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Stop

Function name

`HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)`

Function description

Stops the TIM One Pulse signal generation.

Parameters

- **htim:** pointer to a `TIM_HandleTypeDef` structure that contains the configuration information for TIM module.
- **OutputChannel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Start_IT

Function name

`HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)`

Function description

Starts the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** pointer to a `TIM_HandleTypeDef` structure that contains the configuration information for TIM module.
- **OutputChannel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Stop_IT

Function name

`HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)`

Function description

Stops the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **OutputChannel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Init

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)

Function description

Initialize the TIM Encoder Interface and initialize the associated handle.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **sConfig:** TIM Encoder Interface configuration structure.

Return values

- **HAL:** status

HAL_TIM_Encoder_DelInit

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_DelInit (TIM_HandleTypeDef * htim)

Function description

DeInitialize the TIM Encoder interface.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** status

HAL_TIM_Encoder_MspInit

Function name

void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)

Function description

Initialize the TIM Encoder Interface MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_Encoder_MspDeInit

Function name

```
void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)
```

Function description

DeInitialize TIM Encoder Interface MSP.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_Encoder_Start

Function name

```
HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Starts the TIM Encoder Interface.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop

Function name

```
HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Stops the TIM Encoder Interface.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_IT

Function name

```
HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Starts the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)

Function description

Starts the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
- **pData1:** The destination Buffer address for Input Capture Channel1.
- **pData2:** The destination Buffer address for Input Capture Channel2.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_IRQHandler

Function name

void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)

Function description

This function handles TIM interrupts requests.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initialize the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **sConfig:** TIM Output Compare configuration structure.
- **Channel:** : TIM Channels to configure. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
 - TIM_CHANNEL_5: TIM Channel 5 selected
 - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_ConfigChannel

Function name

```
HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)
```

Function description

Initialize the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **sConfig:** TIM PWM configuration structure.
- **Channel:** : TIM Channels to be configured. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
 - TIM_CHANNEL_5: TIM Channel 5 selected
 - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_IC_ConfigChannel

Function name

```
HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)
```

Function description

Initialize the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **sConfig:** TIM Input Capture configuration structure.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_ConfigChannel

Function name

```
HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)
```

Function description

Initialize the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **sConfig:** TIM One Pulse configuration structure.
- **OutputChannel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
- **InputChannel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIM_ConfigOCrefClear

Function name

```
HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim,  
TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)
```

Function description

Configures the OCRef clear feature.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel. This parameter can be one of the following values:
 - TIM_Channel_1: TIM Channel 1
 - TIM_Channel_2: TIM Channel 2
 - TIM_Channel_3: TIM Channel 3
 - TIM_Channel_4: TIM Channel 4
 - TIM_Channel_5: TIM Channel 5
 - TIM_Channel_6: TIM Channel 6

Return values

- **None:**

HAL_TIM_ConfigClockSource

Function name

```
HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef  
* sClockSourceConfig)
```

Function description

Configures the clock source to be used.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **sClockSourceConfig:** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.

Return values

- **HAL:** status

HAL_TIM_ConfigTI1Input

Function name

HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)

Function description

Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **TI1_Selection:** Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values:
 - **TIM_TI1SELECTION_CH1:** The TIMx_CH1 pin is connected to TI1 input
 - **TIM_TI1SELECTION_XORCOMBINATION:** The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchronization

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchronization_IT

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode in interrupt mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStart

Function name

```
HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef *htim, uint32_t  
BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t *BurstBuffer, uint32_t BurstLength)
```

Function description

Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **BurstBaseAddress:** TIM Base address from when the DMA will starts the Data write. This parameters can be on of the following values:
 - TIM_DMABASE_CR1
 - TIM_DMABASE_CR2
 - TIM_DMABASE_SMCR
 - TIM_DMABASE_DIER
 - TIM_DMABASE_SR
 - TIM_DMABASE_EGR
 - TIM_DMABASE_CCMR1
 - TIM_DMABASE_CCMR2
 - TIM_DMABASE_CCER
 - TIM_DMABASE_CNT
 - TIM_DMABASE_PSC
 - TIM_DMABASE_ARR
 - TIM_DMABASE_RCR
 - TIM_DMABASE_CCR1
 - TIM_DMABASE_CCR2
 - TIM_DMABASE_CCR3
 - TIM_DMABASE_CCR4
 - TIM_DMABASE_BDTR
 - TIM_DMABASE_DCR
- **BurstRequestSrc:** TIM DMA Request sources. This parameters can be on of the following values:
 - TIM_DMA_UPDATE: TIM update Interrupt source
 - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
 - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
 - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
 - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
 - TIM_DMA_COM: TIM Commutation DMA source
 - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStop

Function name

```
HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef *htim, uint32_t t  
BurstRequestSrc)
```

Function description

Stops the TIM DMA Burst mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **BurstRequestSrc:** TIM DMA Request sources to disable

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStart

Function name

```
HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef *htim, uint32_t t  
BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t *BurstBuffer, uint32_t BurstLength)
```

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **BurstBaseAddress:** TIM Base address from when the DMA will starts the Data read. This parameters can be on of the following values:
 - TIM_DMABASE_CR1
 - TIM_DMABASE_CR2
 - TIM_DMABASE_SMCR
 - TIM_DMABASE_DIER
 - TIM_DMABASE_SR
 - TIM_DMABASE_EGR
 - TIM_DMABASE_CCMR1
 - TIM_DMABASE_CCMR2
 - TIM_DMABASE_CCER
 - TIM_DMABASE_CNT
 - TIM_DMABASE_PSC
 - TIM_DMABASE_ARR
 - TIM_DMABASE_RCR
 - TIM_DMABASE_CCR1
 - TIM_DMABASE_CCR2
 - TIM_DMABASE_CCR3
 - TIM_DMABASE_CCR4
 - TIM_DMABASE_BDTR
 - TIM_DMABASE_DCR
- **BurstRequestSrc:** TIM DMA Request sources. This parameters can be on of the following values:
 - TIM_DMA_UPDATE: TIM update Interrupt source
 - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
 - TIM_DMA_CC2: TIM Capture Compare 2 DMA source

- TIM_DMA_CC3: TIM Capture Compare 3 DMA source
- TIM_DMA_CC4: TIM Capture Compare 4 DMA source
- TIM_DMA_COM: TIM Commutation DMA source
- TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABurstLength_1Transfer and TIM_DMABurstLength_18Transfers.

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStop

Function name

```
HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t  
BurstRequestSrc)
```

Function description

Stop the DMA burst reading.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **BurstRequestSrc:** TIM DMA Request sources to disable.

Return values

- **HAL:** status

HAL_TIM_GenerateEvent

Function name

```
HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)
```

Function description

Generate a software event.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **EventSource:** specifies the event source. This parameter can be one of the following values:
 - TIM_EVENTSOURCE_UPDATE: Timer update Event source
 - TIM_EVENTSOURCE_CC1: TIM Capture Compare 1 Event source
 - TIM_EVENTSOURCE_CC2: TIM Capture Compare 2 Event source
 - TIM_EVENTSOURCE_CC3: TIM Capture Compare 3 Event source
 - TIM_EVENTSOURCE_CC4: TIM Capture Compare 4 Event source
 - TIM_EVENTSOURCE_COM: TIM COM event source
 - TIM_EVENTSOURCE_TRIGGER: TIM Trigger Event source
 - TIM_EVENTSOURCE_BREAK: TIM Break event source
 - TIM_EVENTSOURCE_BREAK2: TIM Break2 event source

Return values

- **HAL:** status

Notes

- TIM6 and TIM7 can only generate an update event.
- TIM_EVENTSOURCE_COM, TIM_EVENTSOURCE_BREAK and TIM_EVENTSOURCE_BREAK2 are used only with TIM1 and TIM8.

HAL_TIM_ReadCapturedValue

Function name

```
uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)
```

Function description

Read the captured value from Capture Compare unit.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** : TIM Channels to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name

```
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)
```

Function description

Period elapsed callback in non blocking mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_OC_DelayElapsedCallback

Function name

```
void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)
```

Function description

Output Compare callback in non blocking mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_IC_CaptureCallback

Function name

```
void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)
```

Function description

Input Capture callback in non blocking mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedCallback

Function name

void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished callback in non blocking mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_TriggerCallback

Function name

void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection callback in non blocking mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_ErrorCallback

Function name

void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)

Function description

TIM error callback in non blocking mode.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

HAL_TIM_Base_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM Base handle state.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** state

HAL_TIM_OC_GetState**Function name**

HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM Output Compare handle state.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** state

HAL_TIM_PWM_GetState**Function name**

HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM PWM handle state.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** state

HAL_TIM_IC_GetState**Function name**

HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM Input Capture handle state.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** state

HAL_TIM_OnePulse_GetState**Function name**

HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM One Pulse Mode handle state.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** state

HAL_TIM_Encoder_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode state.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **HAL:** state

TIM_Base_SetConfig

Function name

void TIM_Base_SetConfig (TIM_TypeDef * TIMx, TIM_Base_InitTypeDef * Structure)

Function description

Time Base configuration.

Parameters

- **TIMx:** TIM peripheral
- **Structure:** TIM Base configuration structure

Return values

- **None:**

TIM_TI1_SetConfig

Function name

void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)

Function description

Configure the TI1 as Input.

Parameters

- **TIMx:** to select the TIM peripheral.
- **TIM_ICPolarity:** : The Input Polarity. This parameter can be one of the following values:
 - **TIM_ICPolarity_Rising**
 - **TIM_ICPolarity_Falling**
 - **TIM_ICPolarity_BothEdge**
- **TIM_ICSelection:** specifies the input to be used. This parameter can be one of the following values:
 - **TIM_ICSelection_DirectTI:** TIM Input 1 is selected to be connected to IC1.
 - **TIM_ICSelection_IndirectTI:** TIM Input 1 is selected to be connected to IC2.
 - **TIM_ICSelection_TRC:** TIM Input 1 is selected to be connected to TRC.
- **TIM_ICFilter:** Specifies the Input Capture Filter. This parameter must be a value between 0x00 and 0x0F.

Return values

- **None:**

Notes

- TIM_ICFilter and TIM_ICPolarity are not used in INDIRECT mode as TI2FP1 (on channel2 path) is used as the input signal. Therefore CCMR1 must be protected against un-initialized filter and polarity values.

TIM_OC1_SetConfig

Function name

```
void TIM_OC1_SetConfig (TIM_TypeDef * TIMx, TIM_OC_InitTypeDef * OC_Config)
```

Function description

TIM Output Compare 1 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The Output configuration structure

Return values

- **None:**

TIM_OC2_SetConfig

Function name

```
void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, TIM_OC_InitTypeDef * OC_Config)
```

Function description

TIM Output Compare 2 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The Output configuration structure

Return values

- **None:**

TIM_OC3_SetConfig

Function name

```
void TIM_OC3_SetConfig (TIM_TypeDef * TIMx, TIM_OC_InitTypeDef * OC_Config)
```

Function description

TIM Output Compare 3 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The output configuration structure

Return values

- **None:**

TIM_OC4_SetConfig

Function name

```
void TIM_OC4_SetConfig (TIM_TypeDef * TIMx, TIM_OC_InitTypeDef * OC_Config)
```

Function description

TIM Output Compare 4 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The Output configuration structure

Return values

- **None:**

TIM_ETR_SetConfig

Function name

```
void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t  
TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)
```

Function description

Configures the TIMx External Trigger (ETR).

Parameters

- **TIMx:** to select the TIM peripheral
- **TIM_ExtTRGPrescaler:** The external Trigger Prescaler. This parameter can be one of the following values:
 - TIM_ETRPRESCALER_DIV1 : ETRP Prescaler OFF.
 - TIM_ETRPRESCALER_DIV2 : ETRP frequency divided by 2.
 - TIM_ETRPRESCALER_DIV4 : ETRP frequency divided by 4.
 - TIM_ETRPRESCALER_DIV8 : ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity:** The external Trigger Polarity. This parameter can be one of the following values:
 - TIM_ETRPOLARITY_INVERTED : active low or falling edge active.
 - TIM_ETRPOLARITY_NONINVERTED : active high or rising edge active.
- **ExtTRGFilter:** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values

- **None:**

HAL_TIM_DMADelayPulseCplt

Function name

```
void HAL_TIM_DMADelayPulseCplt (DMA_HandleTypeDef * hdma)
```

Function description

TIM DMA Delay Pulse complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

HAL_TIM_DMAError

Function name

```
void HAL_TIM_DMAError (DMA_HandleTypeDef * hdma)
```

Function description

TIM DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

HAL_TIM_DMACaptureCplt

Function name

void HAL_TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

TIM_CCxChannelCmd

Function name

void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)

Function description

Enables or disables the TIM Capture Compare Channel x.

Parameters

- **TIMx:** to select the TIM peripheral
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
 - **TIM_Channel_1:** TIM Channel 1
 - **TIM_Channel_2:** TIM Channel 2
 - **TIM_Channel_3:** TIM Channel 3
 - **TIM_Channel_4:** TIM Channel 4
- **ChannelState:** specifies the TIM Channel CCxE bit new state. This parameter can be: **TIM_CCx_ENABLE** or **TIM_CCx_Disable**.

Return values

- **None:**

73.3 TIM Firmware driver defines

73.3.1 TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_ENABLE

TIM_AUTOMATICOUTPUT_DISABLE

TIM Auto-Reload Preload**TIM_AUTORELOAD_PRELOAD_DISABLE**

TIMx_ARR register is not buffered

TIM_AUTORELOAD_PRELOAD_ENABLE

TIMx_ARR register is buffered

TIM Break Input Enable**TIM_BREAK_ENABLE****TIM_BREAK_DISABLE*****TIM Break Input Polarity*****TIM_BREAKPOLARITY_LOW****TIM_BREAKPOLARITY_HIGH*****TIM Clear Input Polarity*****TIM_CLEARINPUTPOLARITY_INVERTED**

Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED

Polarity for ETRx pin

TIM Clear Input Prescaler**TIM_CLEARINPUTPRESCALER_DIV1**

No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Clock Division**TIM_CLOCKDIVISION_DIV1**

Clock Division DIV1

TIM_CLOCKDIVISION_DIV2

Clock Division DIV2

TIM_CLOCKDIVISION_DIV4

Clock Division DIV4

TIM Clock Polarity**TIM_CLOCKPOLARITY_INVERTED**

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_FALLING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE

Polarity for TIx clock sources

TIM Clock Prescaler**TIM_CLOCKPRESCALER_DIV1**

No prescaler is used

TIM_CLOCKPRESCALER_DIV2

Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4

Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8

Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source**TIM_CLOCKSOURCE_ETRMODE2****TIM_CLOCKSOURCE_INTERNAL****TIM_CLOCKSOURCE_ITR0****TIM_CLOCKSOURCE_ITR1****TIM_CLOCKSOURCE_ITR2****TIM_CLOCKSOURCE_ITR3****TIM_CLOCKSOURCE_TI1ED****TIM_CLOCKSOURCE_TI1****TIM_CLOCKSOURCE_TI2****TIM_CLOCKSOURCE_ETRMODE1*****TIM Commutation Source*****TIM_COMMUTATION_TRGI****TIM_COMMUTATION_SOFTWARE*****TIM Counter Mode*****TIM_COUNTERMODE_UP**

Up counting mode

TIM_COUNTERMODE_DOWN

Down counting mode

TIM_COUNTERMODE_CENTERALIGNED1

Center-aligned counting mode 1

TIM_COUNTERMODE_CENTERALIGNED2

Center-aligned counting mode 2

TIM_COUNTERMODE_CENTERALIGNED3

Center-aligned counting mode 3

TIM DMA Base Address**TIM_DMABASE_CR1**

TIM DMA Base Address is CR1

TIM_DMABASE_CR2

TIM DMA Base Address is CR2

TIM_DMABASE_SMCR

TIM DMA Base Address is SMCR

TIM_DMABASE_DIER

TIM DMA Base Address is DIER

TIM_DMABASE_SR

TIM DMA Base Address is SR

TIM_DMABASE_EGR

TIM DMA Base Address is EGR

TIM_DMABASE_CCMR1

TIM DMA Base Address is CCMR1

TIM_DMABASE_CCMR2

TIM DMA Base Address is CCMR2

TIM_DMABASE_CCER

TIM DMA Base Address is CCER

TIM_DMABASE_CNT

TIM DMA Base Address is CNT

TIM_DMABASE_PSC

TIM DMA Base Address is PSC

TIM_DMABASE_ARR

TIM DMA Base Address is ARR

TIM_DMABASE_RCR

TIM DMA Base Address is RCR

TIM_DMABASE_CCR1

TIM DMA Base Address is CCR1

TIM_DMABASE_CCR2

TIM DMA Base Address is CCR2

TIM_DMABASE_CCR3

TIM DMA Base Address is CCR3

TIM_DMABASE_CCR4

TIM DMA Base Address is CCR3

TIM_DMABASE_BDTR

TIM DMA Base Address is BDTR

TIM_DMABASE_DCR

TIM DMA Base Address is DCR

TIM_DMABASE_DMAR

TIM DMA Base Address is DMAR

TIM_DMABASE_AF1

TIM DMA Base Address is AF1

TIM_DMABASE_CCMR3

TIM DMA Base Address is CCMR3

TIM_DMABASE_CCR5

TIM DMA Base Address is CCR5

TIM_DMABASE_CCR6

TIM DMA Base Address is CCR6

TIM_DMABASE_AF2

TIM DMA Base Address is AF2

TIM_DMABASE_AF3

TIM DMA Base Address is AF3

TIM_DMABASE_TISEL

TIM DMA Base Address is TISEL

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER

TIM_DMABURSTLENGTH_2TRANSFERS

TIM_DMABURSTLENGTH_3TRANSFERS

TIM_DMABURSTLENGTH_4TRANSFERS

TIM_DMABURSTLENGTH_5TRANSFERS

TIM_DMABURSTLENGTH_6TRANSFERS

TIM_DMABURSTLENGTH_7TRANSFERS

TIM_DMABURSTLENGTH_8TRANSFERS

TIM_DMABURSTLENGTH_9TRANSFERS
TIM_DMABURSTLENGTH_10TRANSFERS
TIM_DMABURSTLENGTH_11TRANSFERS
TIM_DMABURSTLENGTH_12TRANSFERS
TIM_DMABURSTLENGTH_13TRANSFERS
TIM_DMABURSTLENGTH_14TRANSFERS
TIM_DMABURSTLENGTH_15TRANSFERS
TIM_DMABURSTLENGTH_16TRANSFERS
TIM_DMABURSTLENGTH_17TRANSFERS
TIM_DMABURSTLENGTH_18TRANSFERS

TIM DMA Sources

TIM_DMA_UPDATE
TIM_DMA_CC1
TIM_DMA_CC2
TIM_DMA_CC3
TIM_DMA_CC4
TIM_DMA_COM

TIM_DMA_TRIGGER

TIM Encoder Mode

TIM_ENCODERMODE_TI1
TIM_ENCODERMODE_TI2
TIM_ENCODERMODE_TI12

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED
Polarity for ETR source
TIM_ETRPOLARITY_NONINVERTED
Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1
No prescaler is used

TIM_ETRPRESCALER_DIV2

ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4

ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8

ETR input source is divided by 8

TIM Extended Event Source**TIM_EVENTSOURCE_UPDATE**

Reinitialize the counter and generates an update of the registers

TIM_EVENTSOURCE_CC1

A capture/compare event is generated on channel 1

TIM_EVENTSOURCE_CC2

A capture/compare event is generated on channel 2

TIM_EVENTSOURCE_CC3

A capture/compare event is generated on channel 3

TIM_EVENTSOURCE_CC4

A capture/compare event is generated on channel 4

TIM_EVENTSOURCE_COM

A commutation event is generated

TIM_EVENTSOURCE_TRIGGER

A trigger event is generated

TIM_EVENTSOURCE_BREAK

A break event is generated

TIM_EVENTSOURCE_BREAK2

A break 2 event is generated

TIM_EVENTSOURCE_UPDATE

Reinitialize the counter and generates an update of the registers

TIM_EVENTSOURCE_CC1

A capture/compare event is generated on channel 1

TIM_EVENTSOURCE_CC2

A capture/compare event is generated on channel 2

TIM_EVENTSOURCE_CC3

A capture/compare event is generated on channel 3

TIM_EVENTSOURCE_CC4

A capture/compare event is generated on channel 4

TIM_EVENTSOURCE_COM

A commutation event is generated

TIM_EVENTSOURCE_TRIGGER

A trigger event is generated

TIM_EVENTSOURCE_BREAK

A break event is generated

TIM_EVENTSOURCE_BREAK2

A break 2 event is generated

TIM Exported Macros**__HAL_TIM_RESET_HANDLE_STATE****Description:**

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

__HAL_TIM_ENABLE**Description:**

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

__HAL_TIM_URS_ENABLE**Description:**

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_MOE_ENABLE**Description:**

- Enable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None

TIM_CCER_CCxE_MASK

TIM_CCER_CCxNE_MASK

_HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- _HANDLE_: TIM handle

Return value:

- None

_HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- _HANDLE_: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): _ Counter overflow underflow _ Setting the UG bit _ Update generation through the slave mode controller

_HAL_TIM_MOE_DISABLE

Description:

- Disable the TIM main Output.

Parameters:

- _HANDLE_: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

_HAL_TIM_MOE_DISABLE_UNCONDITIONALLY

Description:

- Disable the TIM main Output.

Parameters:

- _HANDLE_: TIM handle

Return value:

- None

Notes:

- The Main Output Enable of a timer instance is disabled unconditionally

_HAL_TIM_ENABLE_IT

Description:

- Enable the specified TIM interrupt.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__INTERRUPT__`: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
 - `TIM_IT_UPDATE`: Update interrupt
 - `TIM_IT_CC1`: Capture/Compare 1 interrupt
 - `TIM_IT_CC2`: Capture/Compare 2 interrupt
 - `TIM_IT_CC3`: Capture/Compare 3 interrupt
 - `TIM_IT_CC4`: Capture/Compare 4 interrupt
 - `TIM_IT_COM`: Commutation interrupt
 - `TIM_IT_TRIGGER`: Trigger interrupt
 - `TIM_IT_BREAK`: Break interrupt

Return value:

- None

[__HAL_TIM_DISABLE_IT](#)**Description:**

- Disable the specified TIM interrupt.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__INTERRUPT__`: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
 - `TIM_IT_UPDATE`: Update interrupt
 - `TIM_IT_CC1`: Capture/Compare 1 interrupt
 - `TIM_IT_CC2`: Capture/Compare 2 interrupt
 - `TIM_IT_CC3`: Capture/Compare 3 interrupt
 - `TIM_IT_CC4`: Capture/Compare 4 interrupt
 - `TIM_IT_COM`: Commutation interrupt
 - `TIM_IT_TRIGGER`: Trigger interrupt
 - `TIM_IT_BREAK`: Break interrupt

Return value:

- None

[__HAL_TIM_ENABLE_DMA](#)**Description:**

- Enable the specified DMA request.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__DMA__`: specifies the TIM DMA request to enable. This parameter can be one of the following values:
 - `TIM_DMA_UPDATE`: Update DMA request
 - `TIM_DMA_CC1`: Capture/Compare 1 DMA request
 - `TIM_DMA_CC2`: Capture/Compare 2 DMA request
 - `TIM_DMA_CC3`: Capture/Compare 3 DMA request
 - `TIM_DMA_CC4`: Capture/Compare 4 DMA request
 - `TIM_DMA_COM`: Commutation DMA request
 - `TIM_DMA_TRIGGER`: Trigger DMA request

Return value:

- None

[__HAL_TIM_DISABLE_DMA](#)**Description:**

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
 - TIM_DMA_UPDATE: Update DMA request
 - TIM_DMA_CC1: Capture/Compare 1 DMA request
 - TIM_DMA_CC2: Capture/Compare 2 DMA request
 - TIM_DMA_CC3: Capture/Compare 3 DMA request
 - TIM_DMA_CC4: Capture/Compare 4 DMA request
 - TIM_DMA_COM: Commutation DMA request
 - TIM_DMA_TRIGGER: Trigger DMA request
 - TIM_DMA_BREAK: Break DMA request

Return value:

- None

[__HAL_TIM_GET_FLAG](#)**Description:**

- Check whether the specified TIM interrupt flag is set or not.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
 - TIM_FLAG_UPDATE: Update interrupt flag
 - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
 - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
 - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
 - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
 - TIM_FLAG_CC5: Compare 5 interrupt flag
 - TIM_FLAG_CC6: Compare 6 interrupt flag
 - TIM_FLAG_COM: Commutation interrupt flag
 - TIM_FLAG_TRIGGER: Trigger interrupt flag
 - TIM_FLAG_BREAK: Break interrupt flag
 - TIM_FLAG_BREAK2: Break 2 interrupt flag
 - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
 - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
 - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
 - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
 - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

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

[__HAL_TIM_CLEAR_FLAG](#)**Description:**

- Clear the specified TIM interrupt flag.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__FLAG__`: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
 - `TIM_FLAG_UPDATE`: Update interrupt flag
 - `TIM_FLAG_CC1`: Capture/Compare 1 interrupt flag
 - `TIM_FLAG_CC2`: Capture/Compare 2 interrupt flag
 - `TIM_FLAG_CC3`: Capture/Compare 3 interrupt flag
 - `TIM_FLAG_CC4`: Capture/Compare 4 interrupt flag
 - `TIM_FLAG_CC5`: Compare 5 interrupt flag
 - `TIM_FLAG_CC6`: Compare 6 interrupt flag
 - `TIM_FLAG_COM`: Commutation interrupt flag
 - `TIM_FLAG_TRIGGER`: Trigger interrupt flag
 - `TIM_FLAG_BREAK`: Break interrupt flag
 - `TIM_FLAG_BREAK2`: Break 2 interrupt flag
 - `TIM_FLAG_SYSTEM_BREAK`: System Break interrupt flag
 - `TIM_FLAG_CC1OF`: Capture/Compare 1 overcapture flag
 - `TIM_FLAG_CC2OF`: Capture/Compare 2 overcapture flag
 - `TIM_FLAG_CC3OF`: Capture/Compare 3 overcapture flag
 - `TIM_FLAG_CC4OF`: Capture/Compare 4 overcapture flag

Return value:

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

[`__HAL_TIM_GET_IT_SOURCE`](#)**Description:**

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

Parameters:

- `__HANDLE__`: TIM handle
- `__INTERRUPT__`: specifies the TIM interrupt source to check. This parameter can be one of the following values:
 - `TIM_IT_UPDATE`: Update interrupt
 - `TIM_IT_CC1`: Capture/Compare 1 interrupt
 - `TIM_IT_CC2`: Capture/Compare 2 interrupt
 - `TIM_IT_CC3`: Capture/Compare 3 interrupt
 - `TIM_IT_CC4`: Capture/Compare 4 interrupt
 - `TIM_IT_COM`: Commutation interrupt
 - `TIM_IT_TRIGGER`: Trigger interrupt
 - `TIM_IT_BREAK`: Break interrupt

Return value:

- The: state of `TIM_IT` (SET or RESET).

[`__HAL_TIM_CLEAR_IT`](#)**Description:**

- Clear the TIM interrupt pending bits.

Parameters:

- `__HANDLE__`: TIM handle
- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - `TIM_IT_UPDATE`: Update interrupt

- TIM_IT_CC1: Capture/Compare 1 interrupt
- TIM_IT_CC2: Capture/Compare 2 interrupt
- TIM_IT_CC3: Capture/Compare 3 interrupt
- TIM_IT_CC4: Capture/Compare 4 interrupt
- TIM_IT_COM: Commutation interrupt
- TIM_IT_TRIGGER: Trigger interrupt
- TIM_IT_BREAK: Break interrupt

Return value:

- None

[__HAL_TIM_IS_TIM_COUNTING_DOWN](#)**Description:**

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly useful to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

[__HAL_TIM_SET_PRESCALER](#)**Description:**

- Set the TIM Prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __PRESC__: specifies the Prescaler new value.

Return value:

- None

[TIM_SET_ICPRESCALERVALUE](#)[TIM_RESET_ICPRESCALERVALUE](#)[TIM_SET_CAPTUREPOLARITY](#)[TIM_RESET_CAPTUREPOLARITY](#)[__HAL_TIM_SET_COUNTER](#)**Description:**

- Set the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __COUNTER__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_COUNTER

Description:

- Get the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer counter register (TIMx_CNT)

__HAL_TIM_SET_AUTORELOAD

Description:

- Set the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __AUTORELOAD__: specifies the Counter register new value.

Return value:

- None

__HAL_TIM_GET_AUTORELOAD

Description:

- Get the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Set the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __CKD__: specifies the clock division value. This parameter can be one of the following value:
 - TIM_CLOCKDIVISION_DIV1
 - TIM_CLOCKDIVISION_DIV2
 - TIM_CLOCKDIVISION_DIV4

Return value:

- None

__HAL_TIM_GET_CLOCKDIVISION

Description:

- Get the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- The: clock division can be one of the following values:
 - TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT

- TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
- TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

__HAL_TIM_SET_ICPRESCALER

Description:

- Set the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
- __ICPSC__: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
 - TIM_ICPSC_DIV1: no prescaler
 - TIM_ICPSC_DIV2: capture is done once every 2 events
 - TIM_ICPSC_DIV4: capture is done once every 4 events
 - TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

__HAL_TIM_GET_ICPRESCALER

Description:

- Get the TIM Input Capture prescaler on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
 - TIM_CHANNEL_1: get input capture 1 prescaler value
 - TIM_CHANNEL_2: get input capture 2 prescaler value
 - TIM_CHANNEL_3: get input capture 3 prescaler value
 - TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:

- The: input capture prescaler can be one of the following values:
 - TIM_ICPSC_DIV1: no prescaler
 - TIM_ICPSC_DIV2: capture is done once every 2 events
 - TIM_ICPSC_DIV4: capture is done once every 4 events
 - TIM_ICPSC_DIV8: capture is done once every 8 events

__HAL_TIM_SET_CAPTUREPOLARITY

Description:

- Set the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected

- TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
 - TIM_INPUTCHANNELPOLARITY_RISING: Rising Edge
 - TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
 - TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None

[__HAL_TIM_SET_COMPARE](#)**Description:**

- Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
 - TIM_CHANNEL_5: TIM Channel 5 selected
 - TIM_CHANNEL_6: TIM Channel 6 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None

[__HAL_TIM_GET_COMPARE](#)**Description:**

- Get the TIM Capture Compare Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channel associated with the capture compare register This parameter can be one of the following values:
 - TIM_CHANNEL_1: get capture/compare 1 register value
 - TIM_CHANNEL_2: get capture/compare 2 register value
 - TIM_CHANNEL_3: get capture/compare 3 register value
 - TIM_CHANNEL_4: get capture/compare 4 register value
 - TIM_CHANNEL_5: get capture/compare 5 register value
 - TIM_CHANNEL_6: get capture/compare 6 register value

Return value:

- None: by
- 16-bit: or 32-bit value of the capture/compare register (TIMx_CC_Ry)

TIM Flag Definition[TIM_FLAG_UPDATE](#)[TIM_FLAG_CC1](#)[TIM_FLAG_CC2](#)

TIM_FLAG_CC3
TIM_FLAG_CC4
TIM_FLAG_CC5
TIM_FLAG_CC6
TIM_FLAG_COM
TIM_FLAG_TRIGGER
TIM_FLAG_BREAK
TIM_FLAG_BREAK2
TIM_FLAG_SYSTEM_BREAK
TIM_FLAG_CC1OF
TIM_FLAG_CC2OF
TIM_FLAG_CC3OF
TIM_FLAG_CC4OF

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING
TIM_ICPOLARITY_FALLING
TIM_ICPOLARITY_BOTHEDGE

TIM Input Capture Prescaler

TIM_ICPSC_DIV1
Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2
Capture performed once every 2 events

TIM_ICPSC_DIV4
Capture performed once every 4 events

TIM_ICPSC_DIV8
Capture performed once every 8 events

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI
TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

TIM_ICSELECTION_INDIRECTTI
TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

TIM_ICSELECTION_TRC

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM Input Channel polarity**TIM_INPUTCHANNELPOLARITY_RISING**

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_FALLING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE

Polarity for TIx source

TIM interrupt Definition**TIM_IT_UPDATE****TIM_IT_CC1****TIM_IT_CC2****TIM_IT_CC3****TIM_IT_CC4****TIM_IT_COM****TIM_IT_TRIGGER****TIM_IT_BREAK*****TIM Lock Configuration*****TIM_LOCKLEVEL_OFF****TIM_LOCKLEVEL_1****TIM_LOCKLEVEL_2****TIM_LOCKLEVEL_3*****TIM Master Mode Selection*****TIM_TRGO_RESET****TIM_TRGO_ENABLE****TIM_TRGO_UPDATE****TIM_TRGO_OC1****TIM_TRGO_OC1REF****TIM_TRGO_OC2REF****TIM_TRGO_OC3REF**

`TIM_TRGO_OC4REF`

TIM Master Mode Selection 2 (TRGO2)

`TIM_TRGO2_RESET`

`TIM_TRGO2_ENABLE`

`TIM_TRGO2_UPDATE`

`TIM_TRGO2_OC1`

`TIM_TRGO2_OC1REF`

`TIM_TRGO2_OC2REF`

`TIM_TRGO2_OC3REF`

`TIM_TRGO2_OC4REF`

`TIM_TRGO2_OC5REF`

`TIM_TRGO2_OC6REF`

`TIM_TRGO2_OC4REF_RISINGFALLING`

`TIM_TRGO2_OC6REF_RISINGFALLING`

`TIM_TRGO2_OC4REF_RISING_OC6REF_RISING`

`TIM_TRGO2_OC4REF_RISING_OC6REF_FALLING`

`TIM_TRGO2_OC5REF_RISING_OC6REF_RISING`

`TIM_TRGO2_OC5REF_RISING_OC6REF_FALLING`

TIM Master/Slave Mode

`TIM_MASTERSLAVEMODE_ENABLE`

`TIM_MASTERSLAVEMODE_DISABLE`

TIM One Pulse Mode

`TIM_OPMODE_SINGLE`

`TIM_OPMODE_REPETITIVE`

TIM Off-state Selection for Idle Mode

`TIM_OSSI_ENABLE`

`TIM_OSSI_DISABLE`

TIM Off-state Selection for Run Mode

`TIM_OSSR_ENABLE`

`TIM_OSSR_DISABLE`

TIM Output Compare Idle State**TIM_OCIDLESTATE_SET****TIM_OCIDLESTATE_RESET*****TIM Complementary Output Compare Idle State*****TIM_OCNIDLESTATE_SET****TIM_OCNIDLESTATE_RESET*****TIM Complementary Output Compare Polarity*****TIM_OCPOLARITY_HIGH****TIM_OCPOLARITY_LOW*****TIM Output Compare Polarity*****TIM_OCPOLARITY_HIGH****TIM_OCPOLARITY_LOW*****TIM Output Compare State*****TIM_OUTPUTSTATE_DISABLE**

Output State disabled

TIM_OUTPUTSTATE_ENABLE

Output State enabled

TIM Output Fast State**TIM_OCFAST_DISABLE****TIM_OCFAST_ENABLE*****TIM Slave mode*****TIM_SLAVEMODE_DISABLE****TIM_SLAVEMODE_RESET****TIM_SLAVEMODE_GATED****TIM_SLAVEMODE_TRIGGER****TIM_SLAVEMODE_EXTERNAL1****TIM_SLAVEMODE_COMBINED_RESETTRIGGER*****TIM TI1 Input Selection*****TIM_TI1SELECTION_CH1****TIM_TI1SELECTION_XORCOMBINATION*****TIM Trigger Polarity*****TIM_TRIGGERPOLARITY_INVERTED**

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE

Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler**TIM_TRIGGERPRESCALER_DIV1**

No prescaler is used

TIM_TRIGGERPRESCALER_DIV2

Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4

Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8

Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection**TIM_TS_ITR0****TIM_TS_ITR1****TIM_TS_ITR2****TIM_TS_ITR3****TIM_TS_TI1F_ED****TIM_TS_TI1FP1****TIM_TS_TI2FP2****TIM_TS_ETRF****TIM_TS_NONE**

74 HAL TIM Extension Driver

74.1 TIMEx Firmware driver registers structures

74.1.1 TIM_HallSensor_InitTypeDef

Data Fields

- `uint32_t IC1Polarity`
- `uint32_t IC1Prescaler`
- `uint32_t IC1Filter`
- `uint32_t Commutation_Delay`

Field Documentation

- `uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity`

Specifies the active edge of the input signal. This parameter can be a value of **TIM Input Capture Polarity**

- `uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler`

Specifies the Input Capture Prescaler. This parameter can be a value of **TIM Input Capture Prescaler**

- `uint32_t TIM_HallSensor_InitTypeDef::IC1Filter`

Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

- `uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay`

Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

74.1.2 TIMEx_BreakInputConfigTypeDef

Data Fields

- `uint32_t Source`
- `uint32_t Enable`
- `uint32_t Polarity`

Field Documentation

- `uint32_t TIMEx_BreakInputConfigTypeDef::Source`

Specifies the source of the timer break input. This parameter can be a value of **TIM Extended Break input source**

- `uint32_t TIMEx_BreakInputConfigTypeDef::Enable`

Specifies whether or not the break input source is enabled. This parameter can be a value of **TIM Extended Break input source enabling**

- `uint32_t TIMEx_BreakInputConfigTypeDef::Polarity`

Specifies the break input source polarity. This parameter can be a value of **TIM Extended Break input polarity** Not relevant when analog watchdog output of the DFSDM1 used as break input source

74.2 TIMEx Firmware driver API description

74.2.1 TIM Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
 - Output Compare
 - PWM generation (Edge and Center-aligned Mode)
 - One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

74.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
 - Complementary Output Compare : HAL_TIM_OC_MspInit().
 - Complementary PWM generation : HAL_TIM_PWM_MspInit().
 - Complementary One-pulse mode output : HAL_TIM_OnePulse_MspInit().
 - Hall Sensor output : HAL_TIM_HallSensor_MspInit().
2. Initialize the TIM low level resources :
 - a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
 - b. TIM pins configuration
 - Enable the clock for the TIM GPIOs using the following function:
__HAL_RCC_GPIOx_CLK_ENABLE();
 - Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
 - HAL_TIMEx_HallSensor_Init and HAL_TIMEx_ConfigCommutationEvent: to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).
5. Activate the TIM peripheral using one of the start functions:
 - Complementary Output Compare : HAL_TIMEx_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OC_Start_IT().
 - Complementary PWM generation : HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT().
 - Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT().
 - Hall Sensor output : HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

74.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Output channels for OC and PWM mode.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.
- Enable or disable channel grouping

This section contains the following APIs:

- [*HAL_TIMEx_ConfigCommutationEvent*](#)
- [*HAL_TIMEx_ConfigCommutationEvent_IT*](#)
- [*HAL_TIMEx_ConfigCommutationEvent_DMA*](#)
- [*HAL_TIMEx_MasterConfigSynchronization*](#)
- [*HAL_TIMEx_ConfigBreakDeadTime*](#)
- [*HAL_TIMEx_ConfigBreakInput*](#)
- [*HAL_TIMEx_RemapConfig*](#)
- [*HAL_TIMEx_TISelection*](#)
- [*HAL_TIMEx_GroupChannel5*](#)

74.2.4 Extended Callbacks functions

This section provides Extended TIM callback functions:

- TIM Commutation callback
- TIM Break callback

This section contains the following APIs:

- [*HAL_TIMEx_CommutationCallback*](#)
- [*HAL_TIMEx_BreakCallback*](#)

74.2.5 Extended Peripheral State functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [*HAL_TIMEx_HallSensor_GetState*](#)

74.2.6 Detailed description of functions

[**HAL_TIMEx_HallSensor_Init**](#)

Function name

`HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim,
TIM_HallSensor_InitTypeDef * sConfig)`

Function description

Initializes the TIM Hall Sensor Interface and initialize the associated handle.

Parameters

- **htim:** TIM Encoder Interface handle
- **sConfig:** TIM Hall Sensor configuration structure.

Return values

- **HAL:** status

[**HAL_TIMEx_HallSensor_DelInit**](#)

Function name

`HAL_StatusTypeDef HAL_TIMEx_HallSensor_DelInit (TIM_HandleTypeDef * htim)`

Function description

Deinitialize the TIM Hall Sensor interface.

Parameters

- **htim:** TIM Hall Sensor handle.

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_MspInit**Function name****void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)****Function description**

Initializes the TIM Hall Sensor MSP.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_HallSensor_MspDeInit**Function name****void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)****Function description**

DeInitialize TIM Hall Sensor MSP.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_HallSensor_Start**Function name****HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)****Function description**

Starts the TIM Hall Sensor Interface.

Parameters

- **htim:** : TIM Hall Sensor handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop**Function name****HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)****Function description**

Stops the TIM Hall sensor Interface.

Parameters

- **htim:** : TIM Hall Sensor handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** : TIM Hall Sensor handle.

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** : TIM handle.

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** : TIM Hall Sensor handle.
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** : TIM handle.

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** : TIM OC handle
- **Channel:** : TIM Channel to be enabled. This parameter can be one of the following values:

- TIM_CHANNEL_1: TIM Channel 1 selected
- TIM_CHANNEL_2: TIM Channel 2 selected
- TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEEx_OCN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** : TIM Output Compare handle.
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEEx_OCN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEEx_OCN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** : TIM Output Compare handle
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation on the complementary output.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation on the complementary output.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Start the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channel to be enabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** : TIM One Pulse handle
- **OutputChannel:** : TIM Channel to be disabled. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_TIMEx_ConfigCommutationEvent

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence.

Parameters

- **htim:** TIM handle
- **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values:
 - TIM_TS_ITR0: Internal trigger 0 selected
 - TIM_TS_ITR1: Internal trigger 1 selected
 - TIM_TS_ITR2: Internal trigger 2 selected
 - TIM_TS_ITR3: Internal trigger 3 selected
 - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** : the Commutation Event source. This parameter can be one of the following values:
 - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
 - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutationEvent_IT

Function name

```
HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
```

Function description

Configure the TIM commutation event sequence with interrupt.

Parameters

- **htim:** TIM handle
- **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values:
 - TIM_TS_ITR0: Internal trigger 0 selected
 - TIM_TS_ITR1: Internal trigger 1 selected
 - TIM_TS_ITR2: Internal trigger 2 selected
 - TIM_TS_ITR3: Internal trigger 3 selected
 - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** : the Commutation Event source. This parameter can be one of the following values:
 - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
 - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutationEvent_DMA

Function name

```
HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
```

Function description

Configure the TIM commutation event sequence with DMA.

Parameters

- **htim:** TIM handle
- **InputTrigger:** : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values:
 - TIM_TS_ITR0: Internal trigger 0 selected
 - TIM_TS_ITR1: Internal trigger 1 selected

- TIM_TS_ITR2: Internal trigger 2 selected
- TIM_TS_ITR3: Internal trigger 3 selected
- TIM_TS_NONE: No trigger is needed
- **CommutationSource:** : the Commutation Event source. This parameter can be one of the following values:
 - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
 - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- : this function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.
- : The user should configure the DMA in his own software, in This function only the COMDE bit is set.

HAL_TIMEx_MasterConfigSynchronization

Function name

```
HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim,  
TIM_MasterConfigTypeDef * sMasterConfig)
```

Function description

Configures the TIM in master mode.

Parameters

- **htim:** TIM handle.
- **sMasterConfig:** pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values

- **HAL:** status

HAL_TIMEx_ConfigBreakDeadTime

Function name

```
HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim,  
TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)
```

Function description

Configures the Break feature, dead time, Lock level, OSS1/OSSR State and the AOE(automatic output enable).

Parameters

- **htim:** TIM handle
- **sBreakDeadTimeConfig:** pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values

- **HAL:** status

HAL_TIMEx_ConfigBreakInput

Function name

```
HAL_StatusTypeDef HAL_TIMEx_ConfigBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput,  
TIMEx_BreakInputConfigTypeDef * sBreakInputConfig)
```

Function description

Configures the break input source.

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to configure. This parameter can be one of the following values:
 - TIM_BREAKINPUT_BRK: Timer break input
 - TIM_BREAKINPUT_BRK2: Timer break 2 input
- **sBreakInputConfig:** Break input source configuration

Return values

- **HAL:** status

HAL_TIMEx_RemapConfig

Function name

```
HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)
```

Function description

Configures the TIMx Remapping input capabilities.

Parameters

- **htim:** TIM handle.
- **Remap:** specifies the TIM remapping source. For TIM1, the parameter is one of the following values:
 - TIM_TIM1_ETR_GPIO: TIM1_ETR is connected to GPIO
 - TIM_TIM1_ETR_COMP1: TIM1_ETR is connected to COMP1 output
 - TIM_TIM1_ETR_COMP2: TIM1_ETR is connected to COMP2 output
 - TIM_TIM1_ETR_ADC1_AWD1: TIM1_ETR is connected to ADC1 AWD1
 - TIM_TIM1_ETR_ADC1_AWD2: TIM1_ETR is connected to ADC1 AWD2
 - TIM_TIM1_ETR_ADC1_AWD3: TIM1_ETR is connected to ADC1 AWD3
 - TIM_TIM1_ETR_ADC3_AWD1: TIM1_ETR is connected to ADC3 AWD1
 - TIM_TIM1_ETR_ADC3_AWD2: TIM1_ETR is connected to ADC3 AWD2
 - TIM_TIM1_ETR_ADC3_AWD3: TIM1_ETR is connected to ADC3 AWD3:For TIM2, the parameter is one of the following values:
 - TIM_TIM2_ETR_GPIO: TIM2_ETR is connected to GPIO
 - TIM_TIM2_ETR_COMP1: TIM2_ETR is connected to COMP1 output
 - TIM_TIM2_ETR_COMP2: TIM2_ETR is connected to COMP2 output
 - TIM_TIM2_ETR_LSE: TIM2_ETR is connected to LSE
 - TIM_TIM2_ETR_SAI1_FSA: TIM2_ETR is connected to SAI1 FS_A
 - TIM_TIM2_ETR_SAI1_FSB: TIM2_ETR is connected to SAI1 FS_BFor TIM3, the parameter is one of the following values:
 - TIM_TIM3_ETR_GPIO: TIM3_ETR is connected to GPIO
 - TIM_TIM3_ETR_COMP1: TIM3_ETR is connected to COMP1 outputFor TIM5, the parameter is one of the following values:
 - TIM_TIM5_ETR_GPIO: TIM5_ETR is connected to GPIO

- TIM_TIM5_ETR_SAI2_FSA: TIM5_ETR is connected to SAI2 FS_A
- TIM_TIM5_ETR_SAI2_FSB: TIM5_ETR is connected to SAI2 FS_B

For TIM8, the parameter is one of the following values:

- TIM_TIM8_ETR_GPIO: TIM8_ETR is connected to GPIO
- TIM_TIM8_ETR_COMP1: TIM8_ETR is connected to COMP1 output
- TIM_TIM8_ETR_COMP2: TIM8_ETR is connected to COMP2 output
- TIM_TIM8_ETR_ADC2_AWD1: TIM8_ETR is connected to ADC2 AWD1
- TIM_TIM8_ETR_ADC2_AWD2: TIM8_ETR is connected to ADC2 AWD2
- TIM_TIM8_ETR_ADC2_AWD3: TIM8_ETR is connected to ADC2 AWD3
- TIM_TIM8_ETR_ADC3_AWD1: TIM8_ETR is connected to ADC3 AWD1
- TIM_TIM8_ETR_ADC3_AWD2: TIM8_ETR is connected to ADC3 AWD2
- TIM_TIM8_ETR_ADC3_AWD3: TIM8_ETR is connected to ADC3 AWD3

Return values

- **HAL:** status

HAL_TIMEx_TISelection

Function name

```
HAL_StatusTypeDef HAL_TIMEx_TISelection (TIM_HandleTypeDef *htim, uint32_t TISelection, uint32_t Channel)
```

Function description

Configures the TIMx input Selection capabilities.

Parameters

- **htim:** TIM handle.
 - **TISelection:** : parameter of the TIM_TISelectionStruct structure.
 - **Channel:** specifies the channels that will be selected for configuration:
 - TIM_CHANNEL_1: TIM Channel 1
 - TIM_CHANNEL_2: TIM Channel 2
 - TIM_CHANNEL_3: TIM Channel 3
 - TIM_CHANNEL_4: TIM Channel 4
- TISelection parameter of the TIM_TISelectionStruct structure is detailed as follows: For TIM1, the parameter is one of the following values:
- TIM_TIM1_TI1_GPIO: TIM1 TI1 is connected to GPIO
 - TIM_TIM1_TI1_COMP1: TIM1 TI1 is connected to COMP1 output
- For TIM2, the parameter is one of the following values:
- TIM_TIM2_TI4_GPIO: TIM2 TI4 is connected to GPIO
 - TIM_TIM2_TI4_COMP1: TIM2 TI4 is connected to COMP1 output
 - TIM_TIM2_TI4_COMP2: TIM2 TI4 is connected to COMP2 output
 - TIM_TIM2_TI4_COMP1_COMP2: TIM2 TI4 is connected to logical OR between COMP1 and COMP2 output
- For TIM3, the parameter is one of the following values:
- TIM_TIM3_TI1_GPIO: TIM3 TI1 is connected to GPIO
 - TIM_TIM3_TI1_COMP1: TIM3 TI1 is connected to COMP1 output
 - TIM_TIM3_TI1_COMP2: TIM3 TI1 is connected to COMP2 output
 - TIM_TIM3_TI1_COMP1_COMP2: TIM3 TI1 is connected to logical OR between COMP1 and COMP2 output
- For TIM5, the parameter is one of the following values:
- TIM_TIM5_TI1_GPIO: TIM5 TI1 is connected to GPIO

- TIM_TIM5_TI1_CAN_TMP: TIM5 TI1 is connected to CAN TMP
- TIM_TIM5_TI1_CAN_RTP: TIM5 TI1 is connected to CAN RTP

For TIM8, the parameter is one of the following values:

- TIM_TIM8_TI1_GPIO: TIM8 TI1 is connected to GPIO
- TIM_TIM8_TI1_COMP2: TIM8 TI1 is connected to COMP2 output

For TIM15, the parameter is one of the following values:

- TIM_TIM15_TI1_GPIO: TIM15 TI1 is connected to GPIO
- TIM_TIM15_TI1_TIM2: TIM15 TI1 is connected to TIM2 CH1
- TIM_TIM15_TI1_TIM3: TIM15 TI1 is connected to TIM3 CH1
- TIM_TIM15_TI1_TIM4: TIM15 TI1 is connected to TIM4 CH1
- TIM_TIM15_TI1_LSE: TIM15 TI1 is connected to LSE
- TIM_TIM15_TI1_CSI: TIM15 TI1 is connected to CSI
- TIM_TIM15_TI1_MCO2: TIM15 TI1 is connected to MCO2
- TIM_TIM15_TI2_GPIO: TIM15 TI2 is connected to GPIO
- TIM_TIM15_TI2_TIM2: TIM15 TI2 is connected to TIM2 CH2
- TIM_TIM15_TI2_TIM3: TIM15 TI2 is connected to TIM3 CH2
- TIM_TIM15_TI2_TIM4: TIM15 TI2 is connected to TIM4 CH2

For TIM16, the parameter can have the following values:

- TIM_TIM16_TI1_GPIO: TIM16 TI1 is connected to GPIO
- TIM_TIM16_TI1_LSI: TIM16 TI1 is connected to LSI
- TIM_TIM16_TI1_LSE: TIM16 TI1 is connected to LSE
- TIM_TIM16_TI1_RTC: TIM16 TI1 is connected to RTC wakeup interrupt

For TIM17, the parameter can have the following values:

- TIM_TIM17_TI1_GPIO: TIM17 TI1 is connected to GPIO
- TIM_TIM17_TI1_SPDIFFS: TIM17 TI1 is connected to SPDIF FS
- TIM_TIM17_TI1_HSE_1MHZ: TIM17 TI1 is connected to HSE 1MHz
- TIM_TIM17_TI1_MCO1: TIM17 TI1 is connected to MCO1

Return values

- **HAL:** status

HAL_TIMEEx_GroupChannel5

Function name

HAL_StatusTypeDef HAL_TIMEEx_GroupChannel5 (TIM_HandleTypeDef * htim, uint32_t Channels)

Function description

Group channel 5 and channel 1, 2 or 3.

Parameters

- **htim:** TIM handle.
- **Channels:** specifies the reference signal(s) the OC5REF is combined with. This parameter can be any combination of the following values: TIM_GROUPCH5_NONE: No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC TIM_GROUPCH5_OC1REFC: OC1REFC is the logical AND of OC1REFC and OC5REF TIM_GROUPCH5_OC2REFC: OC2REFC is the logical AND of OC2REFC and OC5REF TIM_GROUPCH5_OC3REFC: OC3REFC is the logical AND of OC3REFC and OC5REF

Return values

- **HAL:** status

HAL_TIMEx_CommutationCallback

Function name

```
void HAL_TIMEx_CommutationCallback (TIM_HandleTypeDef * htim)
```

Function description

Hall commutation changed callback in non blocking mode.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

HAL_TIMEx_BreakCallback

Function name

```
void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)
```

Function description

Hall Break detection callback in non blocking mode.

Parameters

- **htim:** : TIM handle

Return values

- **None:**

HAL_TIMEx_HallSensor_GetState

Function name

```
HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (TIM_HandleTypeDef * htim)
```

Function description

Return the TIM Hall Sensor interface state.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** state

HAL_TIMEx_DMAMutationCplt

Function name

```
void HAL_TIMEx_DMAMutationCplt (DMA_HandleTypeDef * hdma)
```

Function description

TIM DMA Commutation callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

74.3 TIMEEx Firmware driver defines

74.3.1 TIMEEx

TIMEX Break input 2 Enable

TIM_BREAK2_DISABLE

TIM Break2 disabled

TIM_BREAK2_ENABLE

TIM Break2 enabled

TIM Extended Break Input 2 Polarity

TIM_BREAK2POLARITY_LOW

TIM Break2 polarity low

TIM_BREAK2POLARITY_HIGH

TIM Break2 polarity high

TIM Extended Break input

TIM_BREAKINPUT_BRK

TIM_BREAKINPUT_BRK2

TIM Extended Break input source

TIM_BREAKINPUTSOURCE_BKIN

TIM_BREAKINPUTSOURCE_COMP1

TIM_BREAKINPUTSOURCE_COMP2

TIM_BREAKINPUTSOURCE_DFSDM1

TIM Extended Break input source enabling

TIM_BREAKINPUTSOURCE_DISABLE

TIM_BREAKINPUTSOURCE_ENABLE

TIM Extended Break input polarity

TIM_BREAKINPUTSOURCE_POLARITY_LOW

TIM_BREAKINPUTSOURCE_POLARITY_HIGH

TIM Extended Channel

TIM_CHANNEL_1

TIM Channel 1

TIM_CHANNEL_2

TIM Channel 2

TIM_CHANNEL_3

TIM Channel 3

TIM_CHANNEL_4

TIM Channel 4

TIM_CHANNEL_5

TIM Channel 5

TIM_CHANNEL_6

TIM Channel 6

TIM_CHANNEL_ALL

TIM all Channels

TIM Extended Clear Input Source**TIM_CLEARINPUTSOURCE_ETR**

TIM Clear input source connected to ETR

TIM_CLEARINPUTSOURCE_OCREFCLR

TIM Clear input source connected to OCREFClear

TIM_CLEARINPUTSOURCE_NONE

TIM Clear input source None

Group Channel 5 and Channel 1, 2 or 3**TIM_GROUPCH5_NONE****TIM_GROUPCH5_OC1REFC****TIM_GROUPCH5_OC2REFC****TIM_GROUPCH5_OC3REFC*****TIM Output Compare and PWM Modes*****TIM_OCMODE_TIMING**

TIM Output timing mode

TIM_OCMODE_ACTIVE

TIM Output Active mode

TIM_OCMODE_INACTIVE

TIM Output Inactive mode

TIM_OCMODE_TOGGLE

TIM Output Toggle mode

TIM_OCMODE_PWM1

TIM PWM mode 1

TIM_OCMODE_PWM2

TIM PWM mode 2

TIM_OCMODE_FORCED_ACTIVE

TIM Forced Active mode

TIM_OCMODE_FORCED_INACTIVE

TIM Forced Inactive mode

TIM_OCMODE_RETRIGERRABLE_OPM1

TIM Retrigerrable OPM mode 1

TIM_OCMODE_RETRIGERRABLE_OPM2

TIM Retrigerrable OPM mode 2

TIM_OCMODE_COMBINED_PWM1

TIM Combined PWM mode 1

TIM_OCMODE_COMBINED_PWM2

TIM Combined PWM mode 2

TIM_OCMODE_ASSYMETRIC_PWM1

TIM Asymetric PWM mode 1

TIM_OCMODE_ASSYMETRIC_PWM2

TIM Asymetric PWM mode 2

TIM Extended Remapping

TIM_TIM1_ETR_GPIO

TIM_TIM1_ETR_ADC1_AWD1

TIM_TIM1_ETR_ADC1_AWD2

TIM_TIM1_ETR_ADC1_AWD3

TIM_TIM1_ETR_ADC3_AWD1

TIM_TIM1_ETR_ADC3_AWD2

TIM_TIM1_ETR_ADC3_AWD3

TIM_TIM1_ETR_COMP1_OUT

TIM_TIM1_ETR_COMP2_OUT

TIM_TIM8_ETR_GPIO

TIM_TIM8_ETR_ADC2_AWD1

TIM_TIM8_ETR_ADC2_AWD2

TIM_TIM8_ETR_ADC2_AWD3

TIM_TIM8_ETR_ADC3_AWD1

TIM_TIM8_ETR_ADC3_AWD2

TIM_TIM8_ETR_ADC3_AWD3

TIM_TIM8_ETR_COMP1_OUT

TIM_TIM8_ETR_COMP2_OUT

TIM_TIM2_ETR_GPIO
TIM_TIM2_ETR_COMP1_OUT
TIM_TIM2_ETR_COMP2_OUT
TIM_TIM2_ETR_RCC_LSE
TIM_TIM2_ETR_SAI1_FSA
TIM_TIM2_ETR_SAI1_FSB
TIM_TIM3_ETR_GPIO
TIM_TIM3_ETR_COMP1_OUT
TIM_TIM5_ETR_GPIO
TIM_TIM5_ETR_SAI2_FSA
TIM_TIM5_ETR_SAI2_FSB
TIM_TIM1_BKR_GPIO
TIM_TIM1_BKR_COMP1_OUT
TIM_TIM1_BKR_COMP2_OUT
TIM_TIM1_BKR_DFSDM_BRK0
TIM_TIM8_BKR_GPIO
TIM_TIM8_BKR_COMP1_OUT
TIM_TIM8_BKR_COMP2_OUT
TIM_TIM8_BKR_DFSDM_BRK2
TIM_TIM15_BKR_GPIO
TIM_TIM15_BKR_COMP1_OUT
TIM_TIM15_BKR_COMP2_OUT
TIM_TIM15_BKR_DFSDM_BRK0
TIM_TIM16_BKR_GPIO
TIM_TIM16_BKR_COMP1_OUT
TIM_TIM16_BKR_COMP2_OUT
TIM_TIM16_BKR_DFSDM_BRK1
TIM_TIM17_BKR_GPIO

TIM_TIM17_BKR_COMP1_OUT
TIM_TIM17_BKR_COMP2_OUT
TIM_TIM17_BKR_DFSDM_BRK2
TIM_TIM1_BKR2_GPIO
TIM_TIM1_BKR2_COMP1_OUT
TIM_TIM1_BKR2_COMP2_OUT
TIM_TIM1_BKR2_DFSDM_BRK1
TIM_TIM8_BKR2_GPIO
TIM_TIM8_BKR2_COMP1_OUT
TIM_TIM8_BKR2_COMP2_OUT
TIM_TIM8_BKR2_DFSDM_BRK3
TIM_TIM1_TI1_GPIO
TIM_TIM1_TI1_COMP1_OUT
TIM_TIM8_TI1_GPIO
TIM_TIM8_TI1_COMP2_OUT
TIM_TIM2_TI4_GPIO
TIM_TIM2_TI4_COMP1_OUT
TIM_TIM2_TI4_COMP2_OUT
TIM_TIM2_TI4_COMP1COMP2_OUT
TIM_TIM3_TI1_GPIO
TIM_TIM3_TI1_COMP1_OUT
TIM_TIM3_TI1_COMP2_OUT
TIM_TIM3_TI1_COMP1COMP2_OUT
TIM_TIM5_TI1_GPIO
TIM_TIM5_TI1_CAN_TMP
TIM_TIM5_TI1_CAN_RTP
TIM_TIM15_TI1_GPIO
TIM_TIM15_TI1_TIM2_CH1

TIM_TIM15_TI1_TIM3_CH1

TIM_TIM15_TI1_TIM4_CH1

TIM_TIM15_TI1_RCC_LSE

TIM_TIM15_TI1_RCC_CSI

TIM_TIM15_TI1_RCC_MCO2

TIM_TIM15_TI2_GPIO

TIM_TIM15_TI2_TIM2_CH2

TIM_TIM15_TI2_TIM3_CH2

TIM_TIM15_TI2_TIM4_CH2

TIM_TIM16_TI1_GPIO

TIM_TIM16_TI1_RCC_LSI

TIM_TIM16_TI1_RCC_LSE

TIM_TIM16_TI1_WKUP_IT

TIM_TIM17_TI1_GPIO

TIM_TIM17_TI1_SPDIF_FS

TIM_TIM17_TI1_RCC_HSE1MHZ

TIM_TIM17_TI1_RCC_MCO1

TIM Trigger Selection

TIM_TS_ITR4

TIM Internal trigger 4

TIM_TS_ITR5

TIM Internal trigger 5

TIM_TS_ITR6

TIM Internal trigger 6

TIM_TS_ITR7

TIM Internal trigger 7

TIM_TS_ITR8

TIM Internal trigger 8

75 HAL UART Generic Driver

75.1 UART Firmware driver registers structures

75.1.1 **UART_InitTypeDef**

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t StopBits*
- *uint32_t Parity*
- *uint32_t Mode*
- *uint32_t HwFlowCtl*
- *uint32_t OverSampling*
- *uint32_t OneBitSampling*
- *uint32_t Prescaler*
- *uint32_t FIFOMode*
- *uint32_t TXFIFOThreshold*
- *uint32_t RXFIFOThreshold*

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***

This member configures the UART communication baud rate. The baud rate register is computed using the following formula:

- If oversampling is 16 or in LIN mode, Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))
- If oversampling is 8, Baud Rate Register[15:4] = ((2 * PCLKx) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * PCLKx) / ((huart->Init.BaudRate)))[3:0]) >> 1

- ***uint32_t UART_InitTypeDef::WordLength***

Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of **UART Word Length**.

- ***uint32_t UART_InitTypeDef::StopBits***

Specifies the number of stop bits transmitted. This parameter can be a value of **UART Number of Stop Bits**.

- ***uint32_t UART_InitTypeDef::Parity***

Specifies the parity mode. This parameter can be a value of **UART Parity**

Note:

- When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).

- ***uint32_t UART_InitTypeDef::Mode***

Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of **UART Transfer Mode**.

- ***uint32_t UART_InitTypeDef::HwFlowCtl***

Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of **UART Hardware Flow Control**.

- ***uint32_t UART_InitTypeDef::OverSampling***

Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of **UART Over Sampling**.

- `uint32_t UART_InitTypeDef::OneBitSampling`

Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of **UART One Bit Sampling Method**.

- `uint32_t UART_InitTypeDef::Prescaler`

Specifies the prescaler value used to divide the UART clock source. This parameter can be a value of **UART Prescaler**.

- `uint32_t UART_InitTypeDef::FIFOMode`

Specifies if the FIFO mode will be used. This parameter can be a value of **UART FIFO mode**.

- `uint32_t UART_InitTypeDef::TXFIFOThreshold`

Specifies the TXFIFO threshold level. This parameter can be a value of **UART TXFIFO threshold level**.

- `uint32_t UART_InitTypeDef::RXFIFOThreshold`

Specifies the RXFIFO threshold level. This parameter can be a value of **UART RXFIFO threshold level**.

75.1.2 **UART_AdvFeatureInitTypeDef**

Data Fields

- `uint32_t AdvFeatureInit`
- `uint32_t TxPinLevellInvert`
- `uint32_t RxPinLevellInvert`
- `uint32_t DataInvert`
- `uint32_t Swap`
- `uint32_t OverrunDisable`
- `uint32_t DMADisableonRxError`
- `uint32_t AutoBaudRateEnable`
- `uint32_t AutoBaudRateMode`
- `uint32_t MSBFirst`

Field Documentation

- `uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit`

Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of **UART Advanced Feature Initialization Type**.

- `uint32_t UART_AdvFeatureInitTypeDef::TxPinLevellInvert`

Specifies whether the TX pin active level is inverted. This parameter can be a value of **UART Advanced Feature TX Pin Active Level Inversion**.

- `uint32_t UART_AdvFeatureInitTypeDef::RxPinLevellInvert`

Specifies whether the RX pin active level is inverted. This parameter can be a value of **UART Advanced Feature RX Pin Active Level Inversion**.

- `uint32_t UART_AdvFeatureInitTypeDef::DataInvert`

Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of **UART Advanced Feature Binary Data Inversion**.

- `uint32_t UART_AdvFeatureInitTypeDef::Swap`

Specifies whether TX and RX pins are swapped. This parameter can be a value of **UART Advanced Feature RX TX Pins Swap**.

- `uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable`

Specifies whether the reception overrun detection is disabled. This parameter can be a value of **UART Advanced Feature Overrun Disable**.

- `uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError`

Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of **UART Advanced Feature DMA Disable On Rx Error**.

- `uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable`

Specifies whether auto Baud rate detection is enabled. This parameter can be a value of **UART Advanced Feature Auto BaudRate Enable**

- `uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode`

If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of **UART Advanced Feature AutoBaud Rate Mode**.

- `uint32_t UART_AdvFeatureInitTypeDef::MSBFirst`

Specifies whether MSB is sent first on UART line. This parameter can be a value of **UART Advanced Feature MSB First**.

75.1.3 **UART_HandleTypeDef**

Data Fields

- `USART_TypeDef * Instance`
- `UART_InitTypeDef Init`
- `UART_AdvFeatureInitTypeDef AdvancedInit`
- `uint8_t * pTxBuffPtr`
- `uint16_t TxXferSize`
- `__IO uint16_t TxXferCount`
- `uint8_t * pRxBuffPtr`
- `uint16_t RxXferSize`
- `__IO uint16_t RxXferCount`
- `uint16_t Mask`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `__IO HAL_UART_StateTypeDef gState`
- `__IO HAL_UART_StateTypeDef RxState`
- `__IO uint32_t ErrorCode`

Field Documentation

- **`USART_TypeDef* UART_HandleTypeDef::Instance`**
UART registers base address
- **`UART_InitTypeDef UART_HandleTypeDef::Init`**
UART communication parameters
- **`UART_AdvFeatureInitTypeDef UART_HandleTypeDef::AdvancedInit`**
UART Advanced Features initialization parameters
- **`uint8_t* UART_HandleTypeDef::pTxBuffPtr`**
Pointer to UART Tx transfer Buffer
- **`uint16_t UART_HandleTypeDef::TxXferSize`**
UART Tx Transfer size
- **`__IO uint16_t UART_HandleTypeDef::TxXferCount`**

- UART Tx Transfer Counter
- `uint8_t* UART_HandleTypeDef::pRxBuffPtr`
 Pointer to UART Rx transfer Buffer
- `uint16_t UART_HandleTypeDef::RxXferSize`
 UART Rx Transfer size
- `__IO uint16_t UART_HandleTypeDef::RxXferCount`
 UART Rx Transfer Counter
- `uint16_t UART_HandleTypeDef::Mask`
 UART Rx RDR register mask
- `DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx`
 UART Tx DMA Handle parameters
- `DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx`
 UART Rx DMA Handle parameters
- `HAL_LockTypeDef UART_HandleTypeDef::Lock`
 Locking object
- `__IO HAL_UART_StateTypeDef UART_HandleTypeDef::gState`
 UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of `HAL_UART_StateTypeDef`
- `__IO HAL_UART_StateTypeDef UART_HandleTypeDef::RxState`
 UART state information related to Rx operations. This parameter can be a value of `HAL_UART_StateTypeDef`
- `__IO uint32_t UART_HandleTypeDef::ErrorCode`
 UART Error code

75.2 UART Firmware driver API description

75.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a `UART_HandleTypeDef` handle structure (eg. `UART_HandleTypeDef huart`).
2. Initialize the UART low level resources by implementing the `HAL_UART_MspInit()` API:
 - Enable the USARTx interface clock.
 - UART pins configuration:
 - Enable the clock for the UART GPIOs.
 - Configure these UART pins as alternate function pull-up.
 - NVIC configuration if you need to use interrupt process (`HAL_UART_Transmit_IT()` and `HAL_UART_Receive_IT()` APIs):
 - Configure the USARTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
 - UART interrupts handling:

Note:

The specific UART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) are managed using the macros `__HAL_UART_ENABLE_IT()` and `__HAL_UART_DISABLE_IT()` inside the transmit and receive processes.

- DMA Configuration if you need to use DMA process (`HAL_UART_Transmit_DMA()` and `HAL_UART_Receive_DMA()` APIs):

- Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode (Receiver/Transmitter) in the huart handle Init structure.
 4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart handle AdvancedInit structure.
 5. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
 6. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
 7. For the UART LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.
 8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
 9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

Note: These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init()), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

75.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
 - Hardware flow control
 - Receiver/transmitter modes
 - Over Sampling Method
 - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
 - TX and/or RX pin level inversion
 - data logical level inversion
 - RX and TX pins swap
 - RX overrun detection disabling
 - DMA disabling on RX error
 - MSB first on communication line
 - auto Baud rate detection

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init() and HAL_MultiProcessor_Init() API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [**HAL_UART_Init**](#)
- [**HAL_HalfDuplex_Init**](#)

- [*HAL_LIN_Init*](#)
- [*HAL_MultiProcessor_Init*](#)
- [*HAL_UART_DelInit*](#)
- [*HAL_UART_MspInit*](#)
- [*HAL_UART_MspDelInit*](#)

75.2.3 IO operation functions

This section contains the following APIs:

- [*HAL_UART_Transmit*](#)
- [*HAL_UART_Receive*](#)
- [*HAL_UART_Transmit_IT*](#)
- [*HAL_UART_Receive_IT*](#)
- [*HAL_UART_Transmit_DMA*](#)
- [*HAL_UART_Receive_DMA*](#)
- [*HAL_UART_DMAPause*](#)
- [*HAL_UART_DMAResume*](#)
- [*HAL_UART_DMAStop*](#)
- [*HAL_UART_Abort*](#)
- [*HAL_UART_AbortTransmit*](#)
- [*HAL_UART_AbortReceive*](#)
- [*HAL_UART_Abort_IT*](#)
- [*HAL_UART_AbortTransmit_IT*](#)
- [*HAL_UART_AbortReceive_IT*](#)
- [*HAL_UART_IRQHandler*](#)
- [*HAL_UART_TxCpltCallback*](#)
- [*HAL_UART_TxHalfCpltCallback*](#)
- [*HAL_UART_RxCpltCallback*](#)
- [*HAL_UART_RxHalfCpltCallback*](#)
- [*HAL_UART_ErrorCallback*](#)
- [*HAL_UART_AbortCpltCallback*](#)
- [*HAL_UART_AbortTransmitCpltCallback*](#)
- [*HAL_UART_AbortReceiveCpltCallback*](#)

75.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART.

- [*HAL_MultiProcessor_EnableMuteMode\(\)*](#) API enables mute mode
- [*HAL_MultiProcessor_DisableMuteMode\(\)*](#) API disables mute mode
- [*HAL_MultiProcessor_EnterMuteMode\(\)*](#) API enters mute mode
- [*UART_SetConfig\(\)*](#) API configures the UART peripheral
- [*UART_AdvFeatureConfig\(\)*](#) API optionally configures the UART advanced features
- [*UART_CheckIdleState\(\)*](#) API ensures that TEACK and/or REACK are set after initialization
- [*HAL_HalfDuplex_EnableTransmitter\(\)*](#) API disables receiver and enables transmitter
- [*HAL_HalfDuplex_EnableReceiver\(\)*](#) API disables transmitter and enables receiver
- [*HAL_LIN_SendBreak\(\)*](#) API transmits the break characters

This section contains the following APIs:

- [*HAL_MultiProcessor_EnableMuteMode*](#)
- [*HAL_MultiProcessor_DisableMuteMode*](#)
- [*HAL_MultiProcessor_EnterMuteMode*](#)

- [*HAL_HalfDuplex_EnableTransmitter*](#)
- [*HAL_HalfDuplex_EnableReceiver*](#)
- [*HAL_LIN_SendBreak*](#)

75.2.5 Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the UART handle state.
- Return the UART handle error code

This section contains the following APIs:

- [*HAL_UART_GetState*](#)
- [*HAL_UART_GetError*](#)

75.2.6 Detailed description of functions

[**HAL_UART_Init**](#)

Function name

HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)

Function description

Initialize the UART mode according to the specified parameters in the `UART_InitTypeDef` and initialize the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

[**HAL_HalfDuplex_Init**](#)

Function name

HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)

Function description

Initialize the half-duplex mode according to the specified parameters in the `UART_InitTypeDef` and creates the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

[**HAL_LIN_Init**](#)

Function name

HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)

Function description

Initialize the LIN mode according to the specified parameters in the `UART_InitTypeDef` and creates the associated handle .

Parameters

- **huart:** UART handle.
- **BreakDetectLength:** specifies the LIN break detection length. This parameter can be one of the following values:
 - UART_LINBREAKDETECTLENGTH_10B: 10-bit break detection
 - UART_LINBREAKDETECTLENGTH_11B: 11-bit break detection

Return values

- **HAL:** status

HAL_MultiProcessor_Init

Function name

```
HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)
```

Function description

Initialize the multiprocessor mode according to the specified parameters in the `UART_InitTypeDef` and initialize the associated handle.

Parameters

- **huart:** UART handle.
- **Address:** UART node address (4-, 6-, 7- or 8-bit long).
- **WakeUpMethod:** specifies the UART wakeup method. This parameter can be one of the following values:
 - UART_WAKEUPMETHOD_IDLELINE: WakeUp by an idle line detection
 - UART_WAKEUPMETHOD_ADDRESSMARK: WakeUp by an address mark

Return values

- **HAL:** status

Notes

- If the user resorts to idle line detection wake up, the `Address` parameter is useless and ignored by the initialization function.
- If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API `HAL_MultiProcessorEx_AddressLength_Set()` must be called after `HAL_MultiProcessor_Init()`.

HAL_UART_DeInit

Function name

```
HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)
```

Function description

Deinitialize the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_MspInit

Function name

```
void HAL_UART_MspInit (UART_HandleTypeDef * huart)
```

Function description

Initialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_MspInit

Function name

void HAL_UART_MspInit (UART_HandleTypeDef * huart)

Function description

Deinitialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_Transmit

Function name

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer.
- **Size:** Amount of data to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_UART_Receive

Function name

HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_UART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_UART_Receive_IT

Function name

HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

HAL_UART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_UART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

HAL_UART_DMAPause

Function name

HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)

Function description

Pause the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAResume

Function name

HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)

Function description

Resume the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAStop

Function name

HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)

Function description

Stop the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_Abort**Function name**

HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortTransmit**Function name**

HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortReceive**Function name**

HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_Abort_IT

Function name

HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortReceive_IT

Function name

`HAL_StatusTypeDef HAL_UART_AbortReceive_IT (UART_HandleTypeDef * huart)`

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_IRQHandler

Function name

`void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)`

Function description

Handle UART interrupt request.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxHalfCpltCallback

Function name

`void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)`

Function description

Tx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxCpltCallback

Function name

`void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)`

Function description

Tx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxHalfCpltCallback

Function name

void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxCpltCallback

Function name

void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_ErrorCallback

Function name

void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)

Function description

UART error callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortCpltCallback

Function name

void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortTransmitCpltCallback

Function name

void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortReceiveCpltCallback

Function name

void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Receive Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_LIN_SendBreak

Function name

HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)

Function description

Transmit break characters.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)

Function description

Enable UART in mute mode (does not mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_DisableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)

Function description

Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name

void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)

Function description

Enter UART mute mode (means UART actually enters mute mode).

Parameters

- **huart:** UART handle.

Return values

- **None:**

Notes

- To exit from mute mode, HAL_MultiProcessor_DisableMuteMode() API must be called.

HAL_HalfDuplex_EnableTransmitter

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)

Function description

Enable the UART transmitter and disable the UART receiver.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name

`HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)`

Function description

Enable the UART receiver and disable the UART transmitter.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status.

HAL_UART_GetState

Function name

`HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)`

Function description

Return the UART handle state.

Parameters

- **huart:** Pointer to a `UART_HandleTypeDef` structure that contains the configuration information for the specified UART.

Return values

- **HAL:** state

HAL_UART_GetError

Function name

`uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)`

Function description

Return the UART handle error code.

Parameters

- **huart:** Pointer to a `UART_HandleTypeDef` structure that contains the configuration information for the specified UART.

Return values

- **UART:** Error Code

UART_SetConfig

Function name

`HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)`

Function description

Configure the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_CheckIdleState**Function name**

HAL_StatusTypeDef **UART_CheckIdleState** (**UART_HandleTypeDef** * **huart**)

Function description

Check the UART Idle State.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_WaitOnFlagUntilTimeout**Function name**

HAL_StatusTypeDef **UART_WaitOnFlagUntilTimeout** (**UART_HandleTypeDef** * **huart**, **uint32_t** **Flag**,
FlagStatus **Status**, **uint32_t** **Tickstart**, **uint32_t** **Timeout**)

Function description

Handle UART Communication Timeout.

Parameters

- **huart:** UART handle.
- **Flag:** Specifies the UART flag to check
- **Status:** Flag status (SET or RESET)
- **Tickstart:** Tick start value
- **Timeout:** Timeout duration

Return values

- **HAL:** status

UART_AdvFeatureConfig**Function name**

void **UART_AdvFeatureConfig** (**UART_HandleTypeDef** * **huart**)

Function description

Configure the UART peripheral advanced features.

Parameters

- **huart:** UART handle.

Return values

- **None:**

75.3 **UART Firmware driver defines**

75.3.1 **UART**

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT

No advanced feature initialization

UART_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

UART_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

UART_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

UART_ADVFEATURE_SWAP_INIT

TX/RX pins swap

UART_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

UART_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

UART_ADVFEATURE_AUTOBAUDRATE_INIT

Auto Baud rate detection initialization

UART_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

UART Advanced Feature Auto BaudRate Enable**UART_ADVFEATURE_AUTOBAUDRATE_DISABLE**

RX Auto Baud rate detection enable

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE

RX Auto Baud rate detection disable

UART Advanced Feature AutoBaud Rate Mode**UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT**

Auto Baud rate detection on start bit

UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE

Auto Baud rate detection on falling edge

UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFFRAME

Auto Baud rate detection on 0x7F frame detection

UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME

Auto Baud rate detection on 0x55 frame detection

UART Driver Enable Assertion Time LSB Position In CR1 Register**UART_CR1_DEAT_ADDRESS_LSB_POS**

UART Driver Enable assertion time LSB position in CR1 register

UART Driver Enable DeAssertion Time LSB Position In CR1 Register**UART_CR1_DEDT_ADDRESS_LSB_POS**

UART Driver Enable de-assertion time LSB position in CR1 register

UART Address-matching LSB Position In CR2 Register**UART_CR2_ADDRESS_LSB_POS**

UART address-matching LSB position in CR2 register

UART Advanced Feature Binary Data Inversion**UART_ADVFEATURE_DATAINV_DISABLE**

Binary data inversion disable

UART_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

UART Advanced Feature DMA Disable On Rx Error**UART_ADVFEATURE_DMA_ENABLEONRXERROR**

DMA enable on Reception Error

UART_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

UART DMA Rx**UART_DMA_RX_DISABLE**

UART DMA RX disabled

UART_DMA_RX_ENABLE

UART DMA RX enabled

UART DMA Tx**UART_DMA_TX_DISABLE**

UART DMA TX disabled

UART_DMA_TX_ENABLE

UART DMA TX enabled

UART DriverEnable Polarity**UART_DE_POLARITY_HIGH**

Driver enable signal is active high

UART_DE_POLARITY_LOW

Driver enable signal is active low

UART Exported Macros**_HAL_UART_RESET_HANDLE_STATE****Description:**

- Reset UART handle states.

Parameters:

- __HANDLE__: UART handle.

Return value:

- None

_HAL_UART_FLUSH_DRREGISTER**Description:**

- Flush the UART Data registers.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_CLEAR_FLAG`

Description:

- Clear the specified UART pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be any combination of the following values:
 - `UART_FLAG_WUF`: Wake up from stop mode flag
 - `UART_FLAG_CMF`: Character match flag
 - `UART_FLAG_RTOF`: Receiver timeout flag
 - `UART_FLAG_CTS`: CTS Change flag (not available for UART4 and UART5)
 - `UART_FLAG_LBD`: LIN Break detection flag
 - `UART_FLAG_TC`: Transmission Complete flag
 - `UART_FLAG_TXFE`: TXFIFO Empty flag
 - `UART_FLAG_IDLE`: Idle Line detection flag
 - `UART_FLAG_ORE`: OverRun Error flag
 - `UART_FLAG_NE`: Noise Error flag
 - `UART_FLAG_FE`: Framing Error flag
 - `UART_FLAG_PE`: Parity Error flag

Return value:

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

`__HAL_UART_CLEAR_PFLAG`

Description:

- Clear the UART PE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_CLEAR_FEFLAG`

Description:

- Clear the UART FE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_UART_CLEAR_NEFLAG`

Description:

- Clear the UART NE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

[__HAL_UART_CLEAR_OREFLAG](#)**Description:**

- Clear the UART ORE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

[__HAL_UART_CLEAR_IDLEFLAG](#)**Description:**

- Clear the UART IDLE pending flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

[__HAL_UART_CLEAR_TXFECF](#)**Description:**

- Clear the UART TX FIFO empty clear flag.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

[__HAL_UART_GET_FLAG](#)**Description:**

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

Parameters:

- `__HANDLE__`: specifies the UART Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `UART_FLAG_TXFT`: TXFIFO threshold flag
 - `UART_FLAG_RXFT`: RXFIFO threshold flag
 - `UART_FLAG_RXFF`: RXFIFO Full flag
 - `UART_FLAG_TXFE`: TXFIFO Empty flag
 - `UART_FLAG_RXEACK`: Receive enable acknowledge flag
 - `UART_FLAG_TEACK`: Transmit enable acknowledge flag
 - `UART_FLAG_WUF`: Wake up from stop mode flag
 - `UART_FLAG_RXWU`: Receiver wake up flag (if the UART in mute mode)
 - `UART_FLAG_SBKF`: Send Break flag
 - `UART_FLAG_CMF`: Character match flag
 - `UART_FLAG_BUSY`: Busy flag

- UART_FLAG_ABRF: Auto Baud rate detection flag
- UART_FLAG_ABRE: Auto Baud rate detection error flag
- UART_FLAG_RTOF: Receiver timeout flag
- UART_FLAG_CTS: CTS Change flag
- UART_FLAG_LBD: LIN Break detection flag
- UART_FLAG_TXE: Transmit data register empty flag
- UART_FLAG_TC: Transmission Complete flag
- UART_FLAG_RXNE: Receive data register not empty flag
- UART_FLAG_IDLE: Idle Line detection flag
- UART_FLAG_ORE: OverRun Error flag
- .UART_FLAG_NE: Noise Error flag
- UART_FLAG_FE: Framing Error flag
- UART_FLAG_PE: Parity Error flag

Return value:

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

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
 - UART_IT_RXFF : RXFIFO Full interrupt
 - UART_IT_TXFE : TXFIFO Empty interrupt
 - .UART_IT_RXFT : RXFIFO threshold interrupt
 - UART_IT_TXFT : TXFIFO threshold interrupt
 - UART_IT_WUF: Wakeup from stop mode interrupt
 - UART_IT_CM: Character match interrupt
 - UART_IT_CTS: CTS change interrupt
 - UART_IT_LBD: LIN Break detection interrupt
 - UART_IT_TXE: Transmit Data Register empty interrupt
 - UART_IT_TC: Transmission complete interrupt
 - UART_IT_RXNE: Receive Data register not empty interrupt
 - UART_IT_IDLE: Idle line detection interrupt
 - .UART_IT_PE: Parity Error interrupt
 - UART_IT_ERR: Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_DISABLE_IT

Description:

- Disable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
 - .UART_IT_RXFF : RXFIFO Full interrupt

- UART_IT_TXFE : TXFIFO Empty interrupt
- UART_IT_RXFT : RXFIFO threshold interrupt
- UART_IT_TXFT : TXFIFO threshold interrupt
- UART_IT_WUF: Wakeup from stop mode interrupt
- UART_IT_CM: Character match interrupt
- UART_IT_CTS: CTS change interrupt
- UART_IT_LBD: LIN Break detection interrupt
- UART_IT_TXE: Transmit Data Register empty interrupt
- UART_IT_TC: Transmission complete interrupt
- UART_IT_RXNE: Receive Data register not empty interrupt
- UART_IT_IDLE: Idle line detection interrupt
- UART_IT_PE: Parity Error interrupt
- UART_IT_ERR: Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

[__HAL_UART_GET_IT](#)**Description:**

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT__: specifies the UART interrupt to check. This parameter can be one of the following values:
 - UART_IT_RXFF : RXFIFO Full interrupt
 - UART_IT_TXFE : TXFIFO Empty interrupt
 - UART_IT_RXFT : RXFIFO threshold interrupt
 - UART_IT_TXFT : TXFIFO threshold interrupt
 - UART_IT_WUF: Wakeup from stop mode interrupt
 - UART_IT_CM: Character match interrupt
 - UART_IT_CTS: CTS change interrupt
 - UART_IT_LBD: LIN Break detection interrupt
 - UART_IT_TXE: Transmit Data Register empty interrupt
 - UART_IT_TC: Transmission complete interrupt
 - UART_IT_RXNE: Receive Data register not empty interrupt
 - UART_IT_IDLE: Idle line detection interrupt
 - UART_IT_ORE: OverRun Error interrupt
 - UART_IT_NE: Noise Error interrupt
 - UART_IT_FE: Framing Error interrupt
 - UART_IT_PE: Parity Error interrupt

Return value:

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

[__HAL_UART_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
 - UART_IT_RXFF : RXFIFO Full interrupt

- UART_IT_TXFE : TXFIFO Empty interrupt
- UART_IT_RXFT : RXFIFO threshold interrupt
- UART_IT_TXFT : TXFIFO threshold interrupt
- UART_IT_CTS: CTS change interrupt (not available for UART4 and UART5)
- UART_IT_LBD: LIN Break detection interrupt
- UART_IT_TXE: Transmit Data Register empty interrupt
- UART_IT_TC: Transmission complete interrupt
- UART_IT_RXNE: Receive Data register not empty interrupt
- UART_IT_IDLE: Idle line detection interrupt
- UART_IT_ORE: OverRun Error interrupt
- UART_IT_NE: Noise Error interrupt
- UART_IT_FE: Framing Error interrupt
- UART_IT_PE: Parity Error interrupt

Return value:

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

[__HAL_UART_CLEAR_IT](#)**Description:**

- Clear the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
 - UART_CLEAR_PEF: Parity Error Clear Flag
 - UART_CLEAR_FEF: Framing Error Clear Flag
 - UART_CLEAR_NEF: Noise detected Clear Flag
 - UART_CLEAR_OREF: OverRun Error Clear Flag
 - UART_CLEAR_IDLEF: IDLE line detected Clear Flag
 - UART_CLEAR_TCF: Transmission Complete Clear Flag
 - UART_CLEAR_LBDF: LIN Break Detection Clear Flag
 - UART_CLEAR_CTSF: CTS Interrupt Clear Flag
 - UART_CLEAR_RTOF: Receiver Time Out Clear Flag
 - UART_CLEAR_CMF: Character Match Clear Flag
 - .UART_CLEAR_WUF: Wake Up from stop mode Clear Flag
 - UART_CLEAR_TXFECF: TXFIFO empty Clear Flag

Return value:

- None

[__HAL_UART_SEND_REQ](#)**Description:**

- Set a specific UART request flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
 - UART_AUTOBAUD_REQUEST: Auto-Baud Rate Request
 - UART_SENDBREAK_REQUEST: Send Break Request
 - UART_MUTE_MODE_REQUEST: Mute Mode Request
 - UART_RXDATA_FLUSH_REQUEST: Receive Data flush Request

- UART_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None

[__HAL_UART_ONE_BIT_SAMPLE_ENABLE](#)**Description:**

- Enable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

[__HAL_UART_ONE_BIT_SAMPLE_DISABLE](#)**Description:**

- Disable the UART one bit sample method.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

[__HAL_UART_ENABLE](#)**Description:**

- Enable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

[__HAL_UART_DISABLE](#)**Description:**

- Disable UART.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

[__HAL_UART_HWCONTROL_CTS_ENABLE](#)**Description:**

- Enable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

[__HAL_UART_HWCONTROL_CTS_DISABLE](#)

Description:

- Disable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

[__HAL_UART_HWCONTROL_RTS_ENABLE](#)

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

[__HAL_UART_HWCONTROL_RTS_DISABLE](#)

Description:

- Disable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e.
__HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e.
__HAL_UART_ENABLE(__HANDLE__)).

UART FIFO mode**UART_FIFOMODE_DISABLE**

FIFO mode disable

UART_FIFOMODE_ENABLE

FIFO mode enable

UART Status Flags**UART_FLAG_TXFT**

UART TXFIFO threshold flag

UART_FLAG_RXFT

UART RXFIFO threshold flag

UART_FLAG_RXFF

UART RXFIFO Full flag

UART_FLAG_TXFE

UART TXFIFO Empty flag

UART_FLAG_TEACK

UART receive enable acknowledge flag

UART_FLAG_TEACK

UART transmit enable acknowledge flag

UART_FLAG_WUF

UART wake-up from stop mode flag

UART_FLAG_RWU

UART receiver wake-up from mute mode flag

UART_FLAG_SBKF

UART send break flag

UART_FLAG_CMF

UART character match flag

UART_FLAG_BUSY

UART busy flag

UART_FLAG_ABRF

UART auto Baud rate flag

UART_FLAG_ABRE

UART auto Baud rate error

UART_FLAG_RTOF

UART receiver timeout flag

UART_FLAG_CTS

UART clear to send flag

UART_FLAG_CTSIF

UART clear to send interrupt flag

UART_FLAG_LBDF

UART LIN break detection flag

UART_FLAG_TXE

UART transmit data register empty

UART_FLAG_TXFNF

UART TXFIFO not full

UART_FLAG_TC

UART transmission complete

UART_FLAG_RXNE

UART read data register not empty

UART_FLAG_RXFNE

UART RXFIFO not empty

UART_FLAG_IDLE

UART idle flag

UART_FLAG_ORE

UART overrun error

UART_FLAG_NE

UART noise error

UART_FLAG_FE

UART frame error

UART_FLAG_PE

UART parity error

UART Half Duplex Selection**UART_HALF_DUPLEX_DISABLE**

UART half-duplex disabled

UART_HALF_DUPLEX_ENABLE

UART half-duplex enabled

UART Hardware Flow Control**UART_HWCONTROL_NONE**

No hardware control

UART_HWCONTROL_RTS

Request To Send

UART_HWCONTROL_CTS

Clear To Send

UART_HWCONTROL_RTS_CTS

Request and Clear To Send

UART Interruptions Flag Mask**UART_IT_MASK**

UART interruptions flags mask

UART Interrupts Definition**UART_IT_PE**

UART parity error interruption

UART_IT_TXE

UART transmit data register empty interruption

UART_IT_TC

UART transmission complete interruption

UART_IT_RXNE

UART read data register not empty interruption

UART_IT_IDLE

UART idle interruption

UART_IT_LBD

UART LIN break detection interruption

UART_IT_CTS

UART CTS interruption

UART_IT_CM

UART character match interruption

UART_IT_WUF

UART wake-up from stop mode interruption

UART_IT_RXFF**UART_IT_TXFE****UART_IT_RXFT****UART_IT_TXFT****UART_IT_ERR****UART_IT_ORE****UART_IT_NE**

UART noise error interruption

UART_IT_FE

UART frame error interruption

UART Interruption Clear Flags**UART_CLEAR_PEF**

Parity Error Clear Flag

UART_CLEAR_FEF

Framing Error Clear Flag

UART_CLEAR_NEF

Noise detected Clear Flag

UART_CLEAR_OREF

OverRun Error Clear Flag

UART_CLEAR_IDLEF

IDLE line detected Clear Flag

UART_CLEAR_TXFECF

TXFIFO empty clear flag

UART_CLEAR_TCF

Transmission Complete Clear Flag

UART_CLEAR_LBDF

LIN Break Detection Clear Flag

UART_CLEAR_CTSF

CTS Interrupt Clear Flag

UART_CLEAR_RTOF

Receiver Time Out Clear Flag

UART_CLEAR_CMF

Character Match Clear Flag

UART_CLEAR_WUF

Wake Up from stop mode Clear Flag

UART Local Interconnection Network mode**UART LIN_DISABLE**

Local Interconnect Network disable

UART LIN_ENABLE

Local Interconnect Network enable

UART LIN Break Detection**UART_LINBREAKDETECTLENGTH_10B**

LIN 10-bit break detection length

UART_LINBREAKDETECTLENGTH_11B

LIN 11-bit break detection length

UART Transfer Mode**UART_MODE_RX**

RX mode

UART_MODE_TX

TX mode

UART_MODE_RX_RX

RX and TX mode

UART Advanced Feature MSB First**UART_ADVFEATURE_MSBFIRST_DISABLE**

Most significant bit sent/received first disable

UART_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable**UART_ADVFEATURE_MUTEMODE_DISABLE**

UART mute mode disable

UART_ADVFEATURE_MUTEMODE_ENABLE

UART mute mode enable

UART One Bit Sampling Method**UART_ONE_BIT_SAMPLE_DISABLE**

One-bit sampling disable

UART_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enable

UART Advanced Feature Overrun Disable**UART_ADVFEATURE_OVERRUN_ENABLE**

RX overrun enable

UART_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

UART Over Sampling**UART_OVERSAMPLING_16**

Oversampling by 16

UART_OVERSAMPLING_8

Oversampling by 8

UART Parity**UART_PARITY_NONE**

No parity

UART_PARITY_EVEN

Even parity

UART_PARITY_ODD

Odd parity

UART Prescaler

UART_PRESCALER_DIV1

 UART clock /1

UART_PRESCALER_DIV2

 UART clock /2

UART_PRESCALER_DIV4

 UART clock /4

UART_PRESCALER_DIV6

 UART clock /6

UART_PRESCALER_DIV8

 UART clock /8

UART_PRESCALER_DIV10

 UART clock /10

UART_PRESCALER_DIV12

 UART clock /12

UART_PRESCALER_DIV16

 UART clock /16

UART_PRESCALER_DIV32

 UART clock /32

UART_PRESCALER_DIV64

 UART clock /64

UART_PRESCALER_DIV128

 UART clock /128

UART_PRESCALER_DIV256

 UART clock /256

UART Receiver TimeOut

UART_RECEIVER_TIMEOUT_DISABLE

 UART receiver timeout disable

UART_RECEIVER_TIMEOUT_ENABLE

 UART receiver timeout enable

UART Request Parameters

UART_AUTOBAUD_REQUEST

 Auto-Baud Rate Request

UART_SENDBREAK_REQUEST

 Send Break Request

UART_MUTE_MODE_REQUEST

 Mute Mode Request

UART_RXDATA_FLUSH_REQUEST

 Receive Data flush Request

UART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

UART_RXFIFO threshold level**UART_RXFIFO_THRESHOLD_1_8**

RXFIFO reaches 1/8 of its depth

UART_RXFIFO_THRESHOLD_1_4

RXFIFO reaches 1/4 of its depth

UART_RXFIFO_THRESHOLD_1_2

RXFIFO reaches 1/2 of its depth

UART_RXFIFO_THRESHOLD_3_4

RXFIFO reaches 3/4 of its depth

UART_RXFIFO_THRESHOLD_7_8

RXFIFO reaches 7/8 of its depth

UART_RXFIFO_THRESHOLD_8_8

RXFIFO becomes full

UART Advanced Feature RX Pin Active Level Inversion**UART_ADVFEATURE_RXINV_DISABLE**

RX pin active level inversion disable

UART_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

UART Advanced Feature RX TX Pins Swap**UART_ADVFEATURE_SWAP_DISABLE**

TX/RX pins swap disable

UART_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

UART State**UART_STATE_DISABLE**

UART disabled

UART_STATE_ENABLE

UART enabled

UART Number of Stop Bits**UART_STOPBITS_0_5**

UART frame with 0.5 stop bit

UART_STOPBITS_1

UART frame with 1 stop bit

UART_STOPBITS_1_5

UART frame with 1.5 stop bits

UART_STOPBITS_2

UART frame with 2 stop bits

UART Advanced Feature Stop Mode Enable**UART_ADVFEATURE_STOPMODE_DISABLE**

UART stop mode disable

UART_ADVFEATURE_STOPMODE_ENABLE

UART stop mode enable

UART polling-based communications time-out value**HAL_UART_TIMEOUT_VALUE**

UART polling-based communications time-out value

UART TXFIFO threshold level**UART_TXFIFO_THRESHOLD_1_8**

TXFIFO reaches 1/8 of its depth

UART_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

UART_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

UART_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

UART_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

UART_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

UART Advanced Feature TX Pin Active Level Inversion**UART_ADVFEATURE_TXINV_DISABLE**

TX pin active level inversion disable

UART_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

UART WakeUp From Stop Selection**UART_WAKEUP_ON_ADDRESS**

UART wake-up on address

UART_WAKEUP_ON_STARTBIT

UART wake-up on start bit

UART_WAKEUP_ON_READDATA_NONEMPTY

UART wake-up on receive data register not empty

UART_WAKEUP_ON_RXFIFO_THRESHOLD

UART wake-up when the RXFIFO reaches threshold

UART_WAKEUP_ON_RXFIFO_FULL

UART wake-up when the RXFIFO is full

UART_WAKEUP_ON_TXFIFO_THRESHOLD

UART wake-up when the TXFIFO reaches threshold

UART_WAKEUP_ON_TXFIFO_EMPTY

UART wake-up when the TXFIFO is empty

UART WakeUp Methods**UART_WAKEUPMETHOD_IDLELINE**

UART wake-up on idle line

UART_WAKEUPMETHOD_ADDRESSMARK

UART wake-up on address mark

76 HAL UART Extension Driver

76.1 UARTE Firmware driver registers structures

76.1.1 UART_WakeUpTypeDef

Data Fields

- *uint32_t WakeUpEvent*
- *uint16_t AddressLength*
- *uint8_t Address*

Field Documentation

- *uint32_t UART_WakeUpTypeDef::WakeUpEvent*

Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of **UART WakeUp From Stop Selection**. If set to **UART_WAKEUP_ON_ADDRESS**, the two other fields below must be filled up.

- *uint16_t UART_WakeUpTypeDef::AddressLength*

Specifies whether the address is 4 or 7-bit long. This parameter can be a value of **UART Extended WakeUp Address Length**.

- *uint8_t UART_WakeUpTypeDef::Address*

UART/USART node address (7-bit long max).

76.2 UARTE Firmware driver API description

76.2.1 UART peripheral extended features

76.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
 - Hardware flow control
 - Receiver/transmitter modes
 - Over Sampling Method
 - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
 - TX and/or RX pin level inversion
 - data logical level inversion
 - RX and TX pins swap
 - RX overrun detection disabling
 - DMA disabling on RX error
 - MSB first on communication line

- auto Baud rate detection

The HAL_RS485Ex_Init() API follows the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [**HAL_RS485Ex_Init**](#)

76.2.3 Peripheral Control functions

This section provides the following functions:

- HAL_UARTEx_EnableClockStopMode() API enables the UART clock (HSI or LSE only) during stop mode
- HAL_UARTEx_DisableClockStopMode() API disables the above functionality
- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_StopModeWakeUpSourceConfig() API defines the wake-up from stop mode trigger: address match, Start Bit detection or RXNE bit status.
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- HAL_UARTEx_WakeupCallback() called upon UART wakeup interrupt

This section contains the following APIs:

- [**HAL_MultiProcessorEx_AddressLength_Set**](#)
- [**HAL_UARTEx_StopModeWakeUpSourceConfig**](#)
- [**HAL_UARTEx_EnableStopMode**](#)
- [**HAL_UARTEx_DisableStopMode**](#)
- [**HAL_UARTEx_WakeupCallback**](#)

76.2.4 Detailed description of functions

[**HAL_RS485Ex_Init**](#)

Function name

HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)

Function description

Initialize the RS485 Driver enable feature according to the specified parameters in the **UART_InitTypeDef** and creates the associated handle.

Parameters

- **huart:** UART handle.
- **Polarity:** select the driver enable polarity. This parameter can be one of the following values:
 - **UART_DE_POLARITY_HIGH:** DE signal is active high
 - **UART_DE_POLARITY_LOW:** DE signal is active low
- **AssertionTime:** Driver Enable assertion time: 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate)
- **DeassertionTime:** Driver Enable deassertion time: 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).

Return values

- **HAL:** status

HAL_UARTEx_StopModeWakeUpSourceConfig

Function name

```
HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart,  
UART_WakeUpTypeDef WakeUpSelection)
```

Function description

Set Wakeup from Stop mode interrupt flag selection.

Parameters

- **huart:** UART handle.
- **WakeUpSelection:** address match, Start Bit detection, RXNE bit status or RX/TX FIFO related event. This parameter can be one of the following values:
 - UART_WAKEUP_ON_ADDRESS
 - UART_WAKEUP_ON_STARTBIT
 - UART_WAKEUP_ON_RXADDATA_NONEMPTY
 - UART_WAKEUP_ON_RXFIFO_THRESHOLD
 - UART_WAKEUP_ON_RXFIFO_FULL
 - UART_WAKEUP_ON_TXFIFO_THRESHOLD
 - UART_WAKEUP_ON_TXFIFO_EMPTY

Return values

- **HAL:** status

HAL_UARTEx_EnableStopMode

Function name

```
HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)
```

Function description

Enable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- The UART is able to wake up the MCU from Stop mode as long as UART clock is HSI or LSE.

HAL_UARTEx_DisableStopMode

Function name

```
HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)
```

Function description

Disable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessorEx_AddressLength_Set

Function name

```
HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)
```

Function description

By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection; this API allows to enable longer addresses detection (6-, 7- or 8-bit long).

Parameters

- **huart:** UART handle.
- **AddressLength:** this parameter can be one of the following values:
 - UART_ADDRESS_DETECT_4B: 4-bit long address
 - UART_ADDRESS_DETECT_7B: 6-, 7- or 8-bit long address

Return values

- **HAL:** status

Notes

- Addresses detection lengths are: 6-bit address detection in 7-bit data mode, 7-bit address detection in 8-bit data mode, 8-bit address detection in 9-bit data mode.

HAL_UARTEx_WakeupCallback

Function name

```
void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)
```

Function description

UART wakeup from Stop mode callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

76.3 UARTEx Firmware driver defines

76.3.1 UARTEx

UART Extended WakeUp Address Length

UART_ADDRESS_DETECT_4B

4-bit long wake-up address

UART_ADDRESS_DETECT_7B

7-bit long wake-up address

UART Word Length

UART_WORDLENGTH_7B

7-bit long UART frame

UART_WORDLENGTH_8B

8-bit long UART frame

UART_WORDLENGTH_9B

9-bit long UART frame

77 HAL USART Generic Driver

77.1 USART Firmware driver registers structures

77.1.1 USART_InitTypeDef

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t StopBits*
- *uint32_t Parity*
- *uint32_t Mode*
- *uint32_t CLKPolarity*
- *uint32_t CLKPhase*
- *uint32_t CLKLastBit*
- *uint32_t Prescaler*
- *uint32_t NSS*
- *uint32_t SlaveMode*
- *uint32_t FIFOMode*
- *uint32_t TXFIFOThreshold*
- *uint32_t RXFIFOThreshold*

Field Documentation

- *uint32_t USART_InitTypeDef::BaudRate*

This member configures the Usart communication baud rate. The baud rate is computed using the following formula: Baud Rate Register = ((PCLKx) / ((husart->Init.BaudRate)))

- *uint32_t USART_InitTypeDef::WordLength*

Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of **USARTEx Word Length**

- *uint32_t USART_InitTypeDef::StopBits*

Specifies the number of stop bits transmitted. This parameter can be a value of **USART Number of Stop Bits**

- *uint32_t USART_InitTypeDef::Parity*

Specifies the parity mode. This parameter can be a value of **USART Parity**

Note:

- When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).

- *uint32_t USART_InitTypeDef::Mode*

Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of **USART Mode**

- *uint32_t USART_InitTypeDef::CLKPolarity*

Specifies the steady state of the serial clock. This parameter can be a value of **USART Clock Polarity**

- *uint32_t USART_InitTypeDef::CLKPhase*

- Specifies the clock transition on which the bit capture is made. This parameter can be a value of **USART Clock Phase**
- **uint32_t USART_InitTypeDef::CLKLastBit**
Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of **USART Last Bit**
 - **uint32_t USART_InitTypeDef::Prescaler**
Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of **USART Prescaler**
 - **uint32_t USART_InitTypeDef::NSS**
Specifies whether the NSS signal is managed by hardware (DIS_NSS pin) or by software . This parameter can be a value of **USART Slave Select Management**
 - **uint32_t USART_InitTypeDef::SlaveMode**
Enable/Disable UART SPI Slave Mode. This parameter can be a value of **USART Synchronous Slave mode enable**
 - **uint32_t USART_InitTypeDef::FIFOMode**
Specifies if the FIFO mode will be used. This parameter can be a value of **USART FIFO mode**
 - **uint32_t USART_InitTypeDef::TXFIFOThreshold**
Specifies the TXFIFO threshold level. This parameter can be a value of **USART TXFIFO threshold level**
 - **uint32_t USART_InitTypeDef::RXFIFOThreshold**
Specifies the RXFIFO threshold level. This parameter can be a value of **USART RXFIFO threshold level**

77.1.2 USART_HandleTypeDef

Data Fields

- **USART_TypeDef * Instance**
- **USART_InitTypeDef Init**
- **uint8_t * pTxBuffPtr**
- **uint16_t TxXferSize**
- **_IO uint16_t TxXferCount**
- **uint8_t * pRxBuffPtr**
- **uint16_t RxXferSize**
- **_IO uint16_t RxXferCount**
- **uint16_t Mask**
- **DMA_HandleTypeDef * hdmatx**
- **DMA_HandleTypeDef * hdmarx**
- **HAL_LockTypeDef Lock**
- **_IO HAL_USART_StateTypeDef State**
- **_IO uint32_t ErrorCode**

Field Documentation

- **USART_TypeDef* USART_HandleTypeDef::Instance**
USART registers base address
- **USART_InitTypeDef USART_HandleTypeDef::Init**
USART communication parameters
- **uint8_t* USART_HandleTypeDef::pTxBuffPtr**
Pointer to USART Tx transfer Buffer
- **uint16_t USART_HandleTypeDef::TxXferSize**

- USART Tx Transfer size
 - `__IO uint16_t USART_HandleTypeDef::TxXferCount`
- USART Tx Transfer Counter
 - `uint8_t* USART_HandleTypeDef::pRxBuffPtr`
- Pointer to USART Rx transfer Buffer
 - `uint16_t USART_HandleTypeDef::RxXferSize`
- USART Rx Transfer size
 - `__IO uint16_t USART_HandleTypeDef::RxXferCount`
- USART Rx Transfer Counter
 - `uint16_t USART_HandleTypeDef::Mask`
- USART Rx RDR register mask
 - `DMA_HandleTypeDef* USART_HandleTypeDef::hdmatx`
- USART Tx DMA Handle parameters
 - `DMA_HandleTypeDef* USART_HandleTypeDef::hdmarx`
- USART Rx DMA Handle parameters
 - `HAL_LockTypeDef USART_HandleTypeDef::Lock`
- Locking object
 - `__IO HAL_USART_StateTypeDef USART_HandleTypeDef::State`
- USART communication state
 - `__IO uint32_t USART_HandleTypeDef::ErrorCode`
- USART Error code

77.2 USART Firmware driver API description

77.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure (eg. USART_HandleTypeDef husart).
2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
 - Enable the USARTx interface clock.
 - USART pins configuration:
 - Enable the clock for the USART GPIOs.
 - Configure these USART pins as alternate function pull-up.
 - NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
 - Configure the USARTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
 - USART interrupts handling:

Note:

The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.

- DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA(), HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.

- Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode (Receiver/Transmitter) in the `husart` handle `Init` structure.
 4. Initialize the USART registers by calling the `HAL_USART_Init()` API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized `HAL_USART_MspInit(&husart)` API.

Note:

To configure and enable/disable the USART to wake up the MCU from stop mode, resort to UART API's `HAL_UARTEx_StopModeWakeUpSourceConfig()`, `HAL_UARTEx_EnableStopMode()` and `HAL_UARTEx_DisableStopMode()` in casting the USART handle to UART type `UART_HandleTypeDef`.

77.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
 - USART polarity
 - USART phase
 - USART LastBit
 - Receiver/transmitter modes

The `HAL_USART_Init()` function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- [`HAL_USART_Init`](#)
- [`HAL_USART_DelInit`](#)
- [`HAL_USART_MspInit`](#)
- [`HAL_USART_MspDelInit`](#)

77.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous data transfers.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The `HAL_USART_TxCpltCallback()`, `HAL_USART_RxCpltCallback()` and `HAL_USART_TxRxCpltCallback()` user callbacks will be executed respectively at the end of the transmit or Receive process. The `HAL_USART_ErrorCallback()` user callback will be executed when a communication error is detected
2. Blocking mode API's are :

- HAL_USART_Transmit() in simplex mode
 - HAL_USART_Receive() in full duplex receive only
 - HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
- HAL_USART_Transmit_IT() in simplex mode
 - HAL_USART_Receive_IT() in full duplex receive only
 - HAL_USART_TransmitReceive_IT() in full duplex mode
 - HAL_USART_IRQHandler()
4. No-Blocking mode API's with DMA are :
- HAL_USART_Transmit_DMA() in simplex mode
 - HAL_USART_Receive_DMA() in full duplex receive only
 - HAL_USART_TransmitReceive_DMA() in full duplex mode
 - HAL_USART_DMAPause()
 - HAL_USART_DMAResume()
 - HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non_Blocking mode:
- HAL_USART_TxCpltCallback()
 - HAL_USART_RxCpltCallback()
 - HAL_USART_TxHalfCpltCallback()
 - HAL_USART_RxHalfCpltCallback()
 - HAL_USART_ErrorCallback()
 - HAL_USART_TxRxCpltCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's : (+) HAL_USART_Abort() (+) HAL_USART_Abort_IT()
7. For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided: (+) HAL_USART_AbortCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
(+) Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user. (+) Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output). (#) There are two modes of transfer: (++) Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer. (++) Non-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_USART_ErrorCallback() user callback will be executed when a communication error is detected (#) Blocking mode API's are : (++) HAL_USART_Transmit() in simplex mode (++) HAL_USART_Receive() in full duplex receive only (++) HAL_USART_TransmitReceive() in full duplex mode (#) Non-Blocking mode API's with Interrupt are : (++) HAL_USART_Transmit_IT() in simplex mode (++) HAL_USART_Receive_IT() in full duplex receive only (++) HAL_USART_TransmitReceive_IT() in full duplex mode (++) HAL_USART_IRQHandler() (#) Non-Blocking mode API's with DMA are : (++) HAL_USART_Transmit_DMA() in simplex mode (++) HAL_USART_Receive_DMA() in full duplex receive only (++) HAL_USART_TransmitReceive_DMA() in full duplex mode (++) HAL_USART_DMAResume() (++) HAL_USART_DMAStop() (#) A set of Transfer Complete Callbacks are provided in Non_Blocking mode: (++) HAL_USART_TxCpltCallback() (++) HAL_USART_RxCpltCallback() (++) HAL_USART_TxHalfCpltCallback() (++) HAL_USART_RxHalfCpltCallback() (++) HAL_USART_ErrorCallback() (++) HAL_USART_TxRxCpltCallback() (#) Non-Blocking mode transfers could be aborted using Abort API's :

- HAL_USART_Abort()
- HAL_USART_Abort_IT() (#) For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided:
- HAL_USART_AbortCpltCallback() (#) In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
- Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user.
- Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

This section contains the following APIs:

- [**HAL_USART_Transmit**](#)
- [**HAL_USART_Receive**](#)
- [**HAL_USART_TransmitReceive**](#)
- [**HAL_USART_Transmit_IT**](#)
- [**HAL_USART_Receive_IT**](#)
- [**HAL_USART_TransmitReceive_IT**](#)
- [**HAL_USART_Transmit_DMA**](#)
- [**HAL_USART_Receive_DMA**](#)
- [**HAL_USART_TransmitReceive_DMA**](#)
- [**HAL_USART_DMAPause**](#)
- [**HAL_USART_DMAResume**](#)
- [**HAL_USART_DMAStop**](#)
- [**HAL_USART_Abort**](#)
- [**HAL_USART_Abort_IT**](#)
- [**HAL_USART_IRQHandler**](#)
- [**HAL_USART_TxCpltCallback**](#)
- [**HAL_USART_TxHalfCpltCallback**](#)
- [**HAL_USART_RxCpltCallback**](#)
- [**HAL_USART_RxHalfCpltCallback**](#)
- [**HAL_USART_TxRxCpltCallback**](#)
- [**HAL_USART_ErrorCallback**](#)
- [**HAL_USART_AbortCpltCallback**](#)

77.2.4

Peripheral State and Error functions

This subsection provides functions allowing to :

- Return the USART handle state
- Return the USART handle error code

This section contains the following APIs:

- [**HAL_USART_GetState**](#)
- [**HAL_USART_GetError**](#)

77.2.5 Detailed description of functions

HAL_USART_Init

Function name

`HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * huart)`

Function description

Initialize the USART mode according to the specified parameters in the USART_InitTypeDef and initialize the associated handle.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DeInit

Function name

`HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * huart)`

Function description

DeInitialize the USART peripheral.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_MspInit

Function name

`void HAL_USART_MspInit (USART_HandleTypeDef * huart)`

Function description

Initialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_MspDeInit

Function name

`void HAL_USART_MspDeInit (USART_HandleTypeDef * huart)`

Function description

DeInitialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_Transmit

Function name

```
HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t  
Size, uint32_t Timeout)
```

Function description

Simplex send an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** Pointer to data buffer.
- **Size:** Amount of data to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_USART_Receive

Function name

```
HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t  
Size, uint32_t Timeout)
```

Function description

Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pRxData:** Pointer to data buffer.
- **Size:** Amount of data to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.

HAL_USART_TransmitReceive

Function name

```
HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData,  
uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
```

Function description

Full-Duplex Send and Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer.
- **pRxData:** pointer to RX data buffer.
- **Size:** amount of data to be sent (same amount to be received).

- **Timeout:** Timeout duration.

Return values

- **HAL:** status

HAL_USART_Transmit_IT

Function name

```
HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef *husart, uint8_t *pTxData,  
uint16_t Size)
```

Function description

Send an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_USART_Receive_IT

Function name

```
HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef *husart, uint8_t *pRxData,  
uint16_t Size)
```

Function description

Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.

HAL_USART_TransmitReceive_IT

Function name

```
HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef *husart, uint8_t *  
pTxData, uint8_t *pRxData, uint16_t Size)
```

Function description

Full-Duplex Send and Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer.
- **pRxData:** pointer to RX data buffer.
- **Size:** amount of data to be sent (same amount to be received).

Return values

- **HAL:** status

HAL_USART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_USART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- The USART DMA transmit channel must be configured in order to generate the clock for the slave.

HAL_USART_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit Receive an amount of data in non-blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer.
- **pRxData:** pointer to RX data buffer.
- **Size:** amount of data to be received/sent.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

HAL_USART_DMAPause**Function name**

HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)

Function description

Pause the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAResume**Function name**

HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)

Function description

Resume the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAStop**Function name**

HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)

Function description

Stop the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_Abort**Function name**

HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_USART_Abort_IT

Function name

HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_USART_IRQHandler

Function name

void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)

Function description

Handle USART interrupt request.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxHalfCpltCallback

Function name

void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxCpltCallback

Function name

void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxCpltCallback

Function name

void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxHalfCpltCallback

Function name

void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxRxCpltCallback

Function name

void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx/Rx Transfers completed callback for the non-blocking process.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_ErrorCallback

Function name

void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)

Function description

USART error callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_AbortCpltCallback

Function name

void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)

Function description

USART Abort Complete callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_GetState

Function name

HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)

Function description

Return the USART handle state.

Parameters

- **husart:** : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle state

HAL_USART_GetError

Function name

uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)

Function description

Return the USART error code.

Parameters

- **husart:** : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle Error Code

77.3 USART Firmware driver defines

77.3.1 USART

USART Clock

USART_CLOCK_DISABLE

USART clock disable

USART_CLOCK_ENABLE

USART clock enable

USART Clock Phase

USART_PHASE_1EDGE

USART frame phase on first clock transition

USART_PHASE_2EDGE

USART frame phase on second clock transition

USART Clock Polarity

USART_POLARITY_LOW

USART Clock signal is steady Low

USART_POLARITY_HIGH

USART Clock signal is steady High

USART Exported Macros

_HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- **_HANDLE_**: USART handle.

Return value:

- None

_HAL_USART_GET_FLAG

Description:

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

Parameters:

- **_HANDLE_**: specifies the USART Handle
- **_FLAG_**: specifies the flag to check. This parameter can be one of the following values:
 - USART_FLAG_TXFT: TXFIFO threshold flag
 - USART_FLAG_RXFT: RXFIFO threshold flag

- USART_FLAG_RXFF: RXFIFO Full flag
- USART_FLAG_TXFE: TXFIFO Empty flag
- USART_FLAG_RXACK: Receive enable acknowledge flag
- USART_FLAG_TEACK: Transmit enable acknowledge flag
- USART_FLAG_BUSY: Busy flag
- USART_FLAG_TXE: Transmit data register empty flag
- USART_FLAG_TC: Transmission Complete flag
- USART_FLAG_RXNE: Receive data register not empty flag
- USART_FLAG_IDLE: Idle Line detection flag
- USART_FLAG_ORE: OverRun Error flag
- USART_FLAG_UDR: UnderRun Error flag
- USART_FLAG_NE: Noise Error flag
- USART_FLAG_FE: Framing Error flag
- USART_FLAG_PE: Parity Error flag

Return value:

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

[_HAL_USART_CLEAR_FLAG](#)**Description:**

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
 - USART_FLAG_TXFT: TXFIFO threshold flag
 - USART_FLAG_RXFT: RXFIFO threshold flag
 - USART_FLAG_RXFF: RXFIFO Full flag
 - USART_FLAG_TXFE: TXFIFO Empty flag
 - USART_FLAG_RXACK: Receive enable acknowledge flag
 - USART_FLAG_TEACK: Transmit enable acknowledge flag
 - USART_FLAG_WUF: Wake up from stop mode flag
 - USART_FLAG_RWU: Receiver wake up flag (is the USART in mute mode)
 - USART_FLAG_SBKF: Send Break flag
 - USART_FLAG_CMF: Character match flag
 - USART_FLAG_BUSY: Busy flag
 - USART_FLAG_ABRF: Auto Baud rate detection flag
 - USART_FLAG_ABRE: Auto Baud rate detection error flag
 - USART_FLAG_RTOF: Receiver timeout flag
 - USART_FLAG_LBD: LIN Break detection flag
 - USART_FLAG_TXE: Transmit data register empty flag
 - USART_FLAG_TC: Transmission Complete flag
 - USART_FLAG_RXNE: Receive data register not empty flag
 - USART_FLAG_IDLE: Idle Line detection flag
 - USART_FLAG_ORE: OverRun Error flag
 - USART_FLAG_NE: Noise Error flag
 - USART_FLAG_FE: Framing Error flag
 - USART_FLAG_PE: Parity Error flag

Return value:

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

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
 - USART_IT_RXFF: RXFIFO Full interrupt
 - USART_IT_TXFE: TXFIFO Empty interrupt
 - USART_IT_RXFT: RXFIFO threshold interrupt
 - USART_IT_TXFT: TXFIFO threshold interrupt
 - USART_IT_TXE : Transmit Data Register empty interrupt
 - USART_IT_TC : Transmission complete interrupt
 - USART_IT_RXNE: Receive Data register not empty interrupt
 - USART_IT_IDLE: Idle line detection interrupt
 - USART_IT_PE : Parity Error interrupt
 - USART_IT_ERR : Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_DISABLE_IT

Description:

- Disable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:
 - USART_IT_RXFF: RXFIFO Full interrupt
 - USART_IT_TXFE: TXFIFO Empty interrupt
 - USART_IT_RXFT: RXFIFO threshold interrupt
 - USART_IT_TXFT: TXFIFO threshold interrupt
 - USART_IT_TXE : Transmit Data Register empty interrupt
 - USART_IT_TC : Transmission complete interrupt
 - USART_IT_RXNE: Receive Data register not empty interrupt
 - USART_IT_IDLE: Idle line detection interrupt
 - USART_IT_PE : Parity Error interrupt
 - USART_IT_ERR : Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_GET_IT

Description:

- Check whether the specified USART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
 - USART_IT_RXFF: RXFIFO Full interrupt

- USART_IT_TXFE: TXFIFO Empty interrupt
- USART_IT_RXFT: RXFIFO threshold interrupt
- USART_IT_TXFT: TXFIFO threshold interrupt
- USART_IT_TXE : Transmit Data Register empty interrupt
- USART_IT_TC : Transmission complete interrupt
- USART_IT_RXNE: Receive Data register not empty interrupt
- USART_IT_IDLE: Idle line detection interrupt
- USART_IT_ORE : OverRun Error interrupt
- USART_IT_UDR : UnderRun Error interrupt
- USART_IT_NE : Noise Error interrupt
- USART_IT_FE : Framing Error interrupt
- USART_IT_PE : Parity Error interrupt

Return value:

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

[__HAL_USART_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
 - USART_IT_RXFF: RXFIFO Full interrupt
 - USART_IT_TXFE: TXFIFO Empty interrupt
 - USART_IT_RXFT: RXFIFO threshold interrupt
 - USART_IT_TXFT: TXFIFO threshold interrupt
 - USART_IT_TXE : Transmit Data Register empty interrupt
 - USART_IT_TC : Transmission complete interrupt
 - USART_IT_RXNE: Receive Data register not empty interrupt
 - USART_IT_IDLE: Idle line detection interrupt
 - USART_IT_ORE : OverRun Error interrupt
 - USART_IT_NE : Noise Error interrupt
 - USART_IT_FE : Framing Error interrupt
 - USART_IT_PE : Parity Error interrupt

Return value:

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

[__HAL_USART_CLEAR_IT](#)**Description:**

- Clear the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
 - USART_CLEAR_PEF: Parity Error Clear Flag
 - USART_CLEAR_FEF: Framing Error Clear Flag
 - USART_CLEAR_NEF: Noise detected Clear Flag
 - USART_CLEAR_OREF: OverRun Error Clear Flag
 - USART_CLEAR_IDLEF: IDLE line detected Clear Flag

- USART_CLEAR_TCF: Transmission Complete Clear Flag
- USART_CLEAR_UDRCF: UnderRun Error Clear Flag
- USART_CLEAR_TXFECF: TXFIFO empty Clear Flag

Return value:

- None

[__HAL_USART_CLEAR_PEFLAG](#)

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

[__HAL_USART_CLEAR_FEFLAG](#)

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

[__HAL_USART_CLEAR_NEFLAG](#)

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

[__HAL_USART_CLEAR_OREFLAG](#)

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

[__HAL_USART_CLEAR_IDLEFLAG](#)

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_UDRFLAG

Description:

- Clear the USART UDR pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_TXFECF

Description:

- Clear the USART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_SEND_REQ

Description:

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
 - USART_RXDATA_FLUSH_REQUEST: Receive Data flush Request
 - USART_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the USART one bit sample method.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- `__HANDLE__`: specifies the USART Handle.

Return value:

- None

`_HAL_USART_DISABLE`**Description:**

- Disable USART.

Parameters:

- `__HANDLE__`: specifies the USART Handle.

Return value:

- None

USART FIFO mode**`USART_FIFOMODE_DISABLE`**

FIFO mode disable

`USART_FIFOMODE_ENABLE`

FIFO mode enable

USART Flags**`USART_FLAG_TXFT`**

USART TXFIFO threshold flag

`USART_FLAG_RXFT`

USART RXFIFO threshold flag

`USART_FLAG_RXFF`

USART RXFIFO Fullflag

`USART_FLAG_TXFE`

USART TXFIFO Empty flag

`USART_FLAG_RXACK`

USART receive enable acknowledge flag

`USART_FLAG_TEACK`

USART transmit enable acknowledge flag

`USART_FLAG_BUSY`

USART busy flag

`USART_FLAG_UDR`

USART SPI slave underrun error

`USART_FLAG_LBDF`

USART LIN break detection flag

`USART_FLAG_TXE`

USART transmit data register empty

USART_FLAG_TXFNF

USART TXFIFO not full

USART_FLAG_TC

USART transmission complete

USART_FLAG_RXNE

USART read data register not empty

USART_FLAG_RXFNE

USART RXFIFO not empty

USART_FLAG_IDLE

USART idle flag

USART_FLAG_ORE

USART overrun error

USART_FLAG_NE

USART noise error

USART_FLAG_FE

USART frame error

USART_FLAG_PE

USART parity error

USART Interruption Flags Mask**USART_IT_MASK**

USART interruptions flags mask

USART Interrupts Definition**USART_IT_PE****USART_IT_TXE****USART_IT_TC****USART_IT_RXNE****USART_IT_IDLE****USART_IT_ERR****USART_IT_RXFF****USART_IT_TXFE****USART_IT_RXFT****USART_IT_TXFT****USART_IT_UDR**

USART_IT_ORE

USART_IT_NE

USART_IT_FE

USART Interruption Clear Flags

USART_CLEAR_PEF

Parity Error Clear Flag

USART_CLEAR_FEF

Framing Error Clear Flag

USART_CLEAR_NEF

Noise detected Clear Flag

USART_CLEAR_OREF

OverRun Error Clear Flag

USART_CLEAR_IDLEF

IDLE line detected Clear Flag

USART_CLEAR_TCF

Transmission Complete Clear Flag

USART_CLEAR_UDRCF

UnderRun Error Clear Flag

USART_CLEAR_TXFECF

TXFIFO empty clear flag

USART Last Bit

USART_LASTBIT_DISABLE

USART frame last data bit clock pulse not output to SCLK pin

USART_LASTBIT_ENABLE

USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX

RX mode

USART_MODE_TX

TX mode

USART_MODE_TX_RX

RX and TX mode

USART Over Sampling

USART_OVERSAMPLING_16

Oversampling by 16

USART_OVERSAMPLING_8

Oversampling by 8

USART Parity**USART_PARITY_NONE**

No parity

USART_PARITY EVEN

Even parity

USART_PARITY ODD

Odd parity

USART Prescaler**USART_PRESCALER_DIV1**

USART clock /1

USART_PRESCALER_DIV2

USART clock /2

USART_PRESCALER_DIV4

USART clock /4

USART_PRESCALER_DIV6

USART clock /6

USART_PRESCALER_DIV8

USART clock /8

USART_PRESCALER_DIV10

USART clock /10

USART_PRESCALER_DIV12

USART clock /12

USART_PRESCALER_DIV16

USART clock /16

USART_PRESCALER_DIV32

USART clock /32

USART_PRESCALER_DIV64

USART clock /64

USART_PRESCALER_DIV128

USART clock /128

USART_PRESCALER_DIV256

USART clock /256

USART Request Parameters**USART_RXDATA_FLUSH_REQUEST**

Receive Data flush Request

USART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

USART RXFIFO threshold level

USART_RXFIFO_THRESHOLD_1_8

RXFIFO reaches 1/8 of its depth

USART_RXFIFO_THRESHOLD_1_4

RXFIFO reaches 1/4 of its depth

USART_RXFIFO_THRESHOLD_1_2

RXFIFO reaches 1/2 of its depth

USART_RXFIFO_THRESHOLD_3_4

RXFIFO reaches 3/4 of its depth

USART_RXFIFO_THRESHOLD_7_8

RXFIFO reaches 7/8 of its depth

USART_RXFIFO_THRESHOLD_8_8

RXFIFO becomes full

USART Synchronous Slave mode enable**USART_SLAVEMODE_DISABLE**

USART SPI Slave Mode Enable

USART_SLAVEMODE_ENABLE

USART SPI Slave Mode Disable

USART Slave Select Management**USART_NSS_HW**

USART Hardware NSS management

USART_NSS_SW

USART Software NSS management

USART Number of Stop Bits**USART_STOPBITS_0_5**

USART frame with 0.5 stop bit

USART_STOPBITS_1

USART frame with 1 stop bit

USART_STOPBITS_1_5

USART frame with 1.5 stop bits

USART_STOPBITS_2

USART frame with 2 stop bits

USART TXFIFO threshold level**USART_TXFIFO_THRESHOLD_1_8**

TXFIFO reaches 1/8 of its depth

USART_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

USART_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

USART_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

USART_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

USART_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

78 HAL USART Extension Driver

78.1 USARTEx Firmware driver defines

78.1.1 USARTEx

USARTEx Word Length

USART_WORDLENGTH_7B

7-bit long USART frame

USART_WORDLENGTH_8B

8-bit long USART frame

USART_WORDLENGTH_9B

9-bit long USART frame

79 HAL WWDG Generic Driver

79.1 WWDG Firmware driver registers structures

79.1.1 WWDG_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t Window*
- *uint32_t Counter*
- *uint32_t EWIMode*

Field Documentation

- *uint32_t WWDG_InitTypeDef::Prescaler*

Specifies the prescaler value of the WWWDG. This parameter can be a value of **WWDG Prescaler**

- *uint32_t WWDG_InitTypeDef::Window*

Specifies the WWWDG window value to be compared to the downcounter. This parameter must be a number
Min_Data = 0x40 and Max_Data = 0x7F

- *uint32_t WWDG_InitTypeDef::Counter*

Specifies the WWWDG free-running downcounter value. This parameter must be a number between Min_Data
= 0x40 and Max_Data = 0x7F

- *uint32_t WWDG_InitTypeDef::EWIMode*

Specifies if WWWDG Early Wakeup Interupt is enable or not. This parameter can be a value of **WWDG Early
Wakeup Interrupt Mode**

79.1.2 WWDG_HandleTypeDefDef

Data Fields

- *WWDG_TypeDef * Instance*
- *WWDG_InitTypeDef Init*

Field Documentation

- *WWDG_TypeDef* WWDG_HandleTypeDefDef::Instance*

Register base address

- *WWDG_InitTypeDef WWDG_HandleTypeDefDef::Init*

WWWDG required parameters

79.2 WWDG Firmware driver API description

79.2.1 WWDG specific features

Once enabled the WWWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (T[6:0] downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.

- Once enabled the WWDG cannot be disabled except by a system reset.
- WWDGRST flag in RCC_CSR register informs when a WWDG reset has occurred (check available with `__HAL_RCC_GET_FLAG(RCC_FLAG_WWDGRST)`).
- The WWDG downcounter input clock is derived from the APB clock divided by a programmable prescaler.
- WWDG downcounter clock (Hz) = PCLK1 / (4096 * Prescaler)
- WWDG timeout (ms) = (1000 * (T[5;0] + 1)) / (WWDG downcounter clock) where T[5;0] are the lowest 6 bits of downcounter.
- WWDG Counter refresh is allowed between the following limits :
 - min time (ms) = (1000 * (T[5;0] - Window)) / (WWDG downcounter clock)
 - max time (ms) = (1000 * (T[5;0] - 0x40)) / (WWDG downcounter clock)
- Min-max timeout value @80 MHz(PCLK1): ~51.2 us / ~26.22 ms
- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. When the downcounter reaches the value 0x40, an EWI interrupt is generated and the corresponding interrupt service routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device. In some applications, the EWI interrupt can be used to manage a software system check and/or system recovery/graceful degradation, without generating a WWDG reset. In this case, the corresponding interrupt service routine (ISR) should reload the WWDG counter to avoid the WWDG reset, then trigger the required actions.
Note:When the EWI interrupt cannot be served, e.g. due to a system lock in a higher priority task, the WWDG reset will eventually be generated.
- Debug mode : When the microcontroller enters debug mode (core halted), the WWDG counter either continues to work normally or stops, depending on DBG_WWDG_STOP configuration bit in DBG module, accessible through `__HAL_DBGMCU_FREEZE_WWDG()` and `__HAL_DBGMCU_UNFREEZE_WWDG()` macros

79.2.2

How to use this driver

- Enable WWDG APB1 clock using `__HAL_RCC_WWDG_CLK_ENABLE()`.
- Set the WWDG prescaler, refresh window, counter value and Early Wakeup Interrupt mode using using `HAL_WWDG_Init()` function. This enables WWDG peripheral and the downcounter starts downcounting from given counter value. Init function can be called again to modify all watchdog parameters, however if EWI mode has been set once, it can't be clear until next reset.
- The application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset using `HAL_WWDG_Refresh()` function. This operation must occur only when the counter is lower than the window value already programmed.
- if Early Wakeup Interrupt mode is enable an interrupt is generated when the counter reaches 0x40. User can add his own code in weak function `HAL_WWDG_EarlyWakeUpCallback()`.

WWDG HAL driver macros list

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

- `__HAL_WWDG_GET_IT_SOURCE`: Check the selected WWDG's interrupt source.
- `__HAL_WWDG_GET_FLAG`: Get the selected WWDG's flag status.
- `__HAL_WWDG_CLEAR_FLAG`: Clear the WWDG's pending flags.

79.2.3

Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and start the WWDG according to the specified parameters in the `WWDG_InitTypeDef` of associated handle.
- Initialize the WWDG MSP.

This section contains the following APIs:

- [`HAL_WWDG_Init`](#)
- [`HAL_WWDG_MspInit`](#)

79.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:

- [*HAL_WWDG_Refresh*](#)
- [*HAL_WWDG_IRQHandler*](#)
- [*HAL_WWDG_EarlyWakeupCallback*](#)

79.2.5 Detailed description of functions

[**HAL_WWDG_Init**](#)

Function name

`HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwdg)`

Function description

Initialize the WWDG according to the specified.

Parameters

- **hwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

[**HAL_WWDG_MspInit**](#)

Function name

`void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwdg)`

Function description

Initialize the WWDG MSP.

Parameters

- **hwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

[**HAL_WWDG_Refresh**](#)

Function name

`HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwdg)`

Function description

Refresh the WWDG.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_IRQHandler

Function name

```
void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)
```

Function description

Handle WWDG interrupt request.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_WWDG_EarlyWakeupCallback

Function name

```
void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)
```

Function description

WWDG Early Wakeup callback.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

79.3 WWDG Firmware driver defines

79.3.1 WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE

EWI Disable

WWDG_EWI_ENABLE

EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:

- Enable the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE_IT

Description:

- Enable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
 - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

[_HAL_WWDG_CLEAR_FLAG](#)**Description:**

- Clear the WWDG's pending flags.

Parameters:

- _HANDLE_: WWDG handle
- _FLAG_: specifies the flag to clear. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- None

[_HAL_WWDG_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- _HANDLE_: WWDG Handle.
- _INTERRUPT_: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
 - WWDG_IT_EWI: Early Wakeup Interrupt

Return value:

- state: of _INTERRUPT_ (TRUE or FALSE).

WWDG Flag definition[WWDG_FLAG_EWIF](#)

Early wakeup interrupt flag

WWDG Interrupt definition[WWDG_IT_EWI](#)

Early wakeup interrupt

WWDG Prescaler[WWDG_PRESCALER_1](#)

WWDG counter clock = (PCLK1/4096)/1

[WWDG_PRESCALER_2](#)

WWDG counter clock = (PCLK1/4096)/2

[WWDG_PRESCALER_4](#)

WWDG counter clock = (PCLK1/4096)/4

[WWDG_PRESCALER_8](#)

WWDG counter clock = (PCLK1/4096)/8

[WWDG_PRESCALER_16](#)

WWDG counter clock = (PCLK1/4096)/16

[WWDG_PRESCALER_32](#)

WWDG counter clock = (PCLK1/4096)/32

WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
 - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
 - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, ...). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which STM32H7 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32H7 devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to [Section 3.4 Devices supported by HAL drivers](#).

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32h7xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration...)

A template is provided in the HAL driver folders (stm32h7xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32h7xx_hal.h file has to be included.

What is the difference between `stm32h7xx_hal_ppp.c/h` and `stm32h7xx_hal_ppp_ex.c/h`?

The HAL driver architecture supports common features across STM32 series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (stm32h7xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32h7xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_stm32h7xx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the **PPP_HandleTypeDef *pHandle** structure located in each driver in addition to the Initialization structure

PPP_HandleTypeDef *pHandle is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of **HAL_PPP_MspInit()** and **HAL_PPP_MspDeInit()** functions?

These function are called within HAL_PPP_Init() and HAL_PPP_Deinit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32h7xx_hal_msp.c. A template is provided in the HAL driver folders (stm32h7xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute **__weak**)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use **HAL_Init()** function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,...), configure the SystTick and the NVIC priority grouping and the hardware low level initialization.

The SystTick configuration shall be adjusted by calling **HAL_RCC_ClockConfig()** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling **HAL_IncTick()** function in SysTick ISR and retrieve the value of this variable by calling **HAL_GetTick()** function.

The call **HAL_GetTick()** function is mandatory when using HAL drivers with Polling Process or when using **HAL_Delay()**.

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,...).
2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,...) by calling HAL_PPP_MspInit() instm32h7xx_hal_msp.c
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under PPP_IRQHandler() function in stm32h7xx_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by __weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP_IRQHandler() without calling HAL_PPP_IRQHandler().

Can I use directly the macros defined in [stm32h7xx_hal_ppp.h](#) ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypeDef structure peripheral handler be declared?

PPP_HandleTypeDef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

Revision history

Table 16. Document revision history

Date	Revision	Changes
11-May-2017	1	Initial release.
12-Jan-2018	2	Changed ARM into Arm in the whole document. Updated typical clock configuration sequence in Section 3.12.2.2 System clock initialization . Updated DMA generic and extension drivers, Flash generic and extension drivers, System driver, NAND Flash generic driver, PCD generic and extension drivers, RCC generic and extension drivers, SPDIFRX generic driver, SPI generic and extension drivers.

Contents

1	General information	2
2	Acronyms and definitions	3
3	Overview of HAL drivers	5
3.1	HAL and user-application files	5
3.1.1	HAL driver files	5
3.1.2	User-application files	6
3.2	HAL data structures	7
3.2.1	Peripheral handle structures	7
3.2.2	Initialization and configuration structure	8
3.2.3	Specific process structures	9
3.3	API classification	9
3.4	Devices supported by HAL drivers	10
3.5	HAL driver rules	12
3.5.1	HAL API naming rules	12
3.5.2	HAL general naming rules	13
3.5.3	HAL interrupt handler and callback functions	14
3.6	HAL generic APIs	14
3.7	HAL extension APIs	16
3.7.1	HAL extension model overview	16
3.7.2	HAL extension model cases	16
3.8	File inclusion model	17
3.9	HAL common resources	18
3.10	HAL configuration	19
3.11	HAL system peripheral handling	19
3.11.1	Clock	19
3.11.2	GPIOs	20
3.11.3	Cortex NVIC and SysTick timer	21
3.11.4	PWR	22
3.11.5	EXTI	22
3.11.6	DMA	23

3.12	How to use HAL drivers	24
3.12.1	HAL usage models	24
3.12.2	HAL initialization	25
3.12.3	HAL IO operation process.	28
3.12.4	Timeout and error management	31
4	HAL System Driver	35
4.1	HAL Firmware driver API description.....	35
4.1.1	How to use this driver.....	35
4.1.2	Initialization and de-initialization functions	35
4.1.3	HAL Control functions	35
4.1.4	Detailed description of functions.....	36
4.2	HAL Firmware driver defines	49
4.2.1	HAL	49
5	HAL ADC Generic Driver.....	58
5.1	ADC Firmware driver registers structures.....	58
5.1.1	ADC_OversamplingTypeDef	58
5.1.2	ADC_InitTypeDef	58
5.1.3	ADC_AnalogWDGConfTypeDef	60
5.2	ADC Firmware driver API description	61
5.2.1	ADC specific features.....	61
5.2.2	How to use this driver.....	62
5.2.3	Initialization and deinitialization functions	63
5.2.4	IO operation functions	64
5.2.5	Peripheral Control functions	64
5.2.6	Peripheral state and errors functions	64
5.2.7	Detailed description of functions.....	65
5.3	ADC Firmware driver defines	74
5.3.1	ADC	74
6	HAL ADC Extension Driver.....	85
6.1	ADCEx Firmware driver registers structures.....	85
6.1.1	ADC_InjectionConfigTypeDef	85
6.1.2	ADC_HandleTypeDef.....	85

6.1.3	ADC_InjOversamplingTypeDef	85
6.1.4	ADC_ChannelConfTypeDef	86
6.1.5	ADC_InjectionConfTypeDef	87
6.1.6	ADC_MultiModeTypeDef	89
6.2	ADCEx Firmware driver API description	90
6.2.1	ADC specific features	90
6.2.2	IO operation functions	90
6.2.3	Peripheral Control functions	91
6.2.4	Detailed description of functions	91
6.3	ADCEx Firmware driver defines	101
6.3.1	ADCEx	101
7	HAL CEC Generic Driver	110
7.1	CEC Firmware driver registers structures	110
7.1.1	CEC_InitTypeDef	110
7.1.2	CEC_HandleTypeDef	111
7.2	CEC Firmware driver API description	112
7.2.1	How to use this driver	112
7.2.2	Initialization and Configuration functions	112
7.2.3	IO operation functions	113
7.2.4	Peripheral Control function	113
7.2.5	Detailed description of functions	113
7.3	CEC Firmware driver defines	117
7.3.1	CEC	117
8	HAL COMP Generic Driver	125
8.1	COMP Firmware driver registers structures	125
8.1.1	COMP_InitTypeDef	125
8.1.2	COMP_HandleTypeDef	125
8.2	COMP Firmware driver API description	126
8.2.1	COMP Peripheral features	126
8.2.2	How to use this driver	126
8.2.3	Initialization and de-initialization functions	127
8.2.4	IO operation functions	127

8.2.5	Peripheral Control functions	127
8.2.6	Peripheral State functions	127
8.2.7	Detailed description of functions	128
8.3	COMP Firmware driver defines	131
8.3.1	COMP	131
9	HAL CORTEX Generic Driver	143
9.1	CORTEX Firmware driver registers structures	143
9.1.1	MPU_Region_InitTypeDef	143
9.2	CORTEX Firmware driver API description	144
9.2.1	How to use this driver	144
9.2.2	Initialization and de-initialization functions	144
9.2.3	Peripheral Control functions	145
9.2.4	Detailed description of functions	145
9.3	CORTEX Firmware driver defines	151
9.3.1	CORTEX	151
10	HAL CRC Generic Driver	154
10.1	CRC Firmware driver registers structures	154
10.1.1	CRC_InitTypeDef	154
10.1.2	CRC_HandleTypeDef	155
10.2	CRC Firmware driver API description	155
10.2.1	CRC How to use this driver	155
10.2.2	Initialization and de-initialization functions	155
10.2.3	Peripheral Control functions	156
10.2.4	Peripheral State functions	156
10.2.5	Detailed description of functions	156
10.3	CRC Firmware driver defines	158
10.3.1	CRC	158
11	HAL CRC Extension Driver	161
11.1	CRCEx Firmware driver API description	161
11.1.1	CRC specific features	161
11.1.2	CRC Extended features functions	161

11.1.3	Detailed description of functions	161
11.2	CRCEx Firmware driver defines.	162
11.2.1	CRCEx	162
12	HAL CRYP Generic Driver	164
12.1	CRYP Firmware driver registers structures	164
12.1.1	CRYP_ConfigTypeDef	164
12.1.2	CRYP_HandleTypeDef	164
12.2	CRYP Firmware driver API description.	165
12.2.1	How to use this driver	165
12.2.2	Initialization, de-initialization and Set and Get configuration functions.	167
12.2.3	Encrypt Decrypt functions	168
12.2.4	CRYP IRQ handler management	168
12.2.5	Detailed description of functions	169
12.3	CRYP Firmware driver defines.	174
12.3.1	CRYP	174
13	HAL CRYP Extension Driver	178
13.1	CRYPEx Firmware driver API description.	178
13.1.1	How to use this driver	178
13.1.2	Extended AES processing functions	178
13.1.3	Detailed description of functions	178
14	HAL DAC Generic Driver	180
14.1	DAC Firmware driver registers structures.	180
14.1.1	DAC_HandleTypeDef	180
14.1.2	DAC_SampleAndHoldConfTypeDef	180
14.1.3	DAC_ChannelConfTypeDef	181
14.2	DAC Firmware driver API description	181
14.2.1	DAC Peripheral features	181
14.2.2	How to use this driver	183
14.2.3	Initialization and de-initialization functions	184
14.2.4	IO operation functions.	184
14.2.5	Peripheral Control functions	185
14.2.6	Peripheral State and Errors functions	185

14.2.7	Detailed description of functions	185
14.3	DAC Firmware driver defines	191
14.3.1	DAC	191
15	HAL DAC Extension Driver	196
15.1	DACE Firmware driver API description	196
15.1.1	How to use this driver	196
15.1.2	Extended features functions	196
15.1.3	Peripheral Control functions	196
15.1.4	Detailed description of functions	196
15.2	DACE Firmware driver defines	201
15.2.1	DACE	201
16	HAL DCMI Generic Driver	204
16.1	DCMI Firmware driver registers structures	204
16.1.1	DCMI_CodesInitTypeDef	204
16.1.2	DCMI_InitTypeDef	204
16.1.3	DCMI_HandleTypeDef	205
16.2	DCMI Firmware driver API description	206
16.2.1	How to use this driver	206
16.2.2	Initialization and Configuration functions	206
16.2.3	IO operation functions	207
16.2.4	Peripheral Control functions	207
16.2.5	Peripheral State and Errors functions	207
16.2.6	Detailed description of functions	207
16.3	DCMI Firmware driver defines	212
16.3.1	DCMI	212
17	HAL DFSDM Generic Driver	219
17.1	DFSDM Firmware driver registers structures	219
17.1.1	DFSDM_Channel_OutputClockTypeDef	219
17.1.2	DFSDM_Channel_InputTypeDef	219
17.1.3	DFSDM_Channel_SerialInterfaceTypeDef	219
17.1.4	DFSDM_Channel_AwdTypeDef	220
17.1.5	DFSDM_Channel_InitTypeDef	220

17.1.6	DFSDM_Channel_HandleTypeDef	220
17.1.7	DFSDM_Filter-RegularParamTypeDef	221
17.1.8	DFSDM_Filter-InjectedParamTypeDef	221
17.1.9	DFSDM_Filter-FilterParamTypeDef	222
17.1.10	DFSDM_Filter-InitTypeDef	222
17.1.11	DFSDM_Filter_HandleTypeDef	222
17.1.12	DFSDM_Filter-AwdParamTypeDef	223
17.2	DFSDM Firmware driver API description	224
17.2.1	How to use this driver	224
17.2.2	Channel initialization and de-initialization functions	226
17.2.3	Channel operation functions	226
17.2.4	Channel state function	226
17.2.5	Filter initialization and de-initialization functions	227
17.2.6	Filter control functions	227
17.2.7	Filter operation functions	227
17.2.8	Filter state functions	228
17.2.9	Detailed description of functions	228
17.3	DFSDM Firmware driver defines	246
17.3.1	DFSDM	246
18	HAL DMA2D Generic Driver	251
18.1	DMA2D Firmware driver registers structures	251
18.1.1	DMA2D_ColorTypeDef	251
18.1.2	DMA2D_CLUTCfgTypeDef	251
18.1.3	DMA2D_InitTypeDef	251
18.1.4	DMA2D_LayerCfgTypeDef	252
18.1.5	__DMA2D_HandleTypeDef	253
18.2	DMA2D Firmware driver API description	253
18.2.1	How to use this driver	253
18.2.2	Initialization and Configuration functions	254
18.2.3	IO operation functions	255
18.2.4	Peripheral Control functions	255
18.2.5	Peripheral State and Errors functions	256

18.2.6	Detailed description of functions	256
18.3	DMA2D Firmware driver defines	265
18.3.1	DMA2D	265
19	HAL DMA Generic Driver	273
19.1	DMA Firmware driver registers structures	273
19.1.1	DMA_InitTypeDef	273
19.1.2	__DMA_HandleTypeDef	274
19.2	DMA Firmware driver API description	275
19.2.1	How to use this driver	275
19.2.2	Initialization and de-initialization functions	276
19.2.3	IO operation functions	277
19.2.4	State and Errors functions	277
19.2.5	Detailed description of functions	277
19.3	DMA Firmware driver defines	281
19.3.1	DMA	281
20	HAL DMA Extension Driver	297
20.1	DMAEx Firmware driver registers structures	297
20.1.1	HAL_DMA_MuxSyncConfigTypeDef	297
20.1.2	HAL_DMA_MuxRequestGeneratorConfigTypeDef	297
20.2	DMAEx Firmware driver API description	298
20.2.1	How to use this driver	298
20.2.2	Extended features functions	298
20.2.3	Detailed description of functions	298
20.3	DMAEx Firmware driver defines	301
20.3.1	DMAEx	301
21	HAL ETH Generic Driver	306
21.1	ETH Firmware driver registers structures	306
21.1.1	__ETH_BufferTypeDef	306
21.1.2	ETH_TxDescListTypeDef	306
21.1.3	ETH_TxPacketConfig	306
21.1.4	ETH_RxDescListTypeDef	307

21.1.5	ETH_RxPacketInfo	308
21.1.6	ETH_MACConfigTypeDef	308
21.1.7	ETH_DMAConfigTypeDef	311
21.1.8	ETH_InitTypeDef	312
21.1.9	ETH_HandleTypeDef	313
21.1.10	ETH_MACFilterConfigTypeDef	314
21.1.11	ETH_PowerDownConfigTypeDef	315
21.2	ETH Firmware driver API description	315
21.2.1	How to use this driver	315
21.2.2	Initialization and Configuration functions	316
21.2.3	IO operation functions	316
21.2.4	Peripheral Control functions	317
21.2.5	Peripheral State and Errors functions	317
21.2.6	Detailed description of functions	317
21.3	ETH Firmware driver defines	330
21.3.1	ETH	330
22	HAL ETH Extension Driver	351
22.1	ETHEEx Firmware driver registers structures	351
22.1.1	ETH_RxVLANConfigTypeDef	351
22.1.2	ETH_TxVLANConfigTypeDef	351
22.1.3	ETH_L3FilterConfigTypeDef	352
22.1.4	ETH_L4FilterConfigTypeDef	352
22.2	ETHEEx Firmware driver API description	353
22.2.1	Extended features functions	353
22.2.2	Detailed description of functions	353
22.3	ETHEEx Firmware driver defines	360
22.3.1	ETHEEx	360
23	HAL FDCAN Generic Driver	363
23.1	FDCAN Firmware driver registers structures	363
23.1.1	FDCAN_InitTypeDef	363
23.1.2	FDCAN_ClkCalUnitTypeDef	365
23.1.3	FDCAN_FilterTypeDef	365

23.1.4	FDCAN_TxHeaderTypeDef	366
23.1.5	FDCAN_RxHeaderTypeDef	367
23.1.6	FDCAN_TxEventFifoTypeDef	368
23.1.7	FDCAN_HpMsgStatusTypeDef	369
23.1.8	FDCAN_ProtocolStatusTypeDef	369
23.1.9	FDCAN_ErrorCountersTypeDef	371
23.1.10	FDCAN_TT_ConfigTypeDef	371
23.1.11	FDCAN_TriggerTypeDef	373
23.1.12	FDCAN_TTOperationStatusTypeDef	374
23.1.13	FDCAN_MsgRamAddressTypeDef	375
23.1.14	FDCAN_HandleTypeDef	376
23.2	FDCAN Firmware driver API description	377
23.2.1	How to use this driver	377
23.2.2	Initialization and de-initialization functions	378
23.2.3	Configuration functions	378
23.2.4	Control functions	379
23.2.5	TT Configuration and control functions	380
23.2.6	Interrupts management	381
23.2.7	Callback functions	381
23.2.8	Peripheral State functions	382
23.2.9	Detailed description of functions	382
23.3	FDCAN Firmware driver defines	411
23.3.1	FDCAN	411
24	HAL FLASH Generic Driver	438
24.1	FLASH Firmware driver registers structures	438
24.1.1	FLASH_ProcessTypeDef	438
24.2	FLASH Firmware driver API description	438
24.2.1	FLASH peripheral features	438
24.2.2	How to use this driver	438
24.2.3	Detailed description of functions	439
24.3	FLASH Firmware driver defines	443
24.3.1	FLASH	443

25	HAL FLASH Extension Driver.....	455
25.1	FLASHEx Firmware driver registers structures	455
25.1.1	FLASH_EraseInitTypeDef.....	455
25.1.2	FLASH_OBProgramInitTypeDef.....	455
25.2	FLASHEx Firmware driver API description.....	457
25.2.1	Flash Extension features	457
25.2.2	How to use this driver.....	457
25.2.3	Detailed description of functions.....	457
25.3	FLASHEx Firmware driver defines	460
25.3.1	FLASHEx	460
26	HAL GPIO Generic Driver.....	467
26.1	GPIO Firmware driver registers structures	467
26.1.1	GPIO_InitTypeDef.....	467
26.2	GPIO Firmware driver API description	467
26.2.1	GPIO Peripheral features	467
26.2.2	How to use this driver.....	468
26.2.3	Initialization and de-initialization functions	468
26.2.4	IO operation functions.....	468
26.2.5	Detailed description of functions.....	468
26.3	GPIO Firmware driver defines	471
26.3.1	GPIO	471
27	HAL GPIO Extension Driver.....	479
27.1	GPIOEx Firmware driver defines	479
27.1.1	GPIOEx	479
28	HAL HASH Generic Driver	480
28.1	HASH Firmware driver registers structures	480
28.1.1	HASH_InitTypeDef	480
28.1.2	HASH_HandleTypeDef.....	480
28.2	HASH Firmware driver API description.....	481
28.2.1	How to use this driver	481
28.2.2	Initialization and de-initialization functions	482

28.2.3	Polling mode HASH processing functions	483
28.2.4	Interruption mode HASH processing functions	483
28.2.5	DMA mode HASH processing functions	484
28.2.6	Polling mode HMAC processing functions	484
28.2.7	Interrupt mode HMAC processing functions	484
28.2.8	DMA mode HMAC processing functions	484
28.2.9	Peripheral State methods	485
28.2.10	Detailed description of functions	485
28.3	HASH Firmware driver defines	500
28.3.1	HASH	500
29	HAL HASH Extension Driver	505
29.1	HASHEx Firmware driver API description	505
29.1.1	HASH peripheral extended features	505
29.1.2	Polling mode HASH extended processing functions	505
29.1.3	Interruption mode HASH extended processing functions	506
29.1.4	DMA mode HASH extended processing functionss	506
29.1.5	Polling mode HMAC extended processing functions	506
29.1.6	Interrupt mode HMAC extended processing functions	507
29.1.7	DMA mode HMAC extended processing functions	507
29.1.8	Multi-buffer DMA mode HMAC extended processing functions	507
29.1.9	Detailed description of functions	508
30	HAL HCD Generic Driver	521
30.1	HCD Firmware driver registers structures	521
30.1.1	HCD_HandleTypeDef	521
30.2	HCD Firmware driver API description	521
30.2.1	How to use this driver	521
30.2.2	Initialization and de-initialization functions	522
30.2.3	IO operation functions	522
30.2.4	Peripheral Control functions	522
30.2.5	Peripheral State functions	522
30.2.6	Detailed description of functions	522
30.3	HCD Firmware driver defines	528

30.3.1	HCD	528
31	HAL HRTIM Generic Driver	530
31.1	HRTIM Firmware driver registers structures	530
31.1.1	HRTIM_InitTypeDef	530
31.1.2	HRTIM_TimerParamTypeDef	530
31.1.3	__HRTIM_HandleTypeDef	531
31.1.4	HRTIM_TimeBaseCfgTypeDef	532
31.1.5	HRTIM_SimpleOCChannelCfgTypeDef	532
31.1.6	HRTIM_SimplePWMChannelCfgTypeDef	533
31.1.7	HRTIM_SimpleCaptureChannelCfgTypeDef	533
31.1.8	HRTIM_SimpleOnePulseChannelCfgTypeDef	533
31.1.9	HRTIM_TimerCfgTypeDef	534
31.1.10	HRTIM_CompareCfgTypeDef	536
31.1.11	HRTIM_CaptureCfgTypeDef	536
31.1.12	HRTIM_OutputCfgTypeDef	537
31.1.13	HRTIM_TimerEventFilteringCfgTypeDef	537
31.1.14	HRTIM_DeadTimeCfgTypeDef	538
31.1.15	HRTIM_ChopperModeCfgTypeDef	538
31.1.16	HRTIM_EventCfgTypeDef	539
31.1.17	HRTIM_FaultCfgTypeDef	539
31.1.18	HRTIM_BurstModeCfgTypeDef	540
31.1.19	HRTIM_ADCTriggerCfgTypeDef	540
31.2	HRTIM Firmware driver API description	541
31.2.1	Simple mode v.s. waveform mode	541
31.2.2	How to use this driver	541
31.2.3	Initialization and Time Base Configuration functions	544
31.2.4	Simple time base mode functions	544
31.2.5	Simple output compare functions	545
31.2.6	Simple PWM output functions	545
31.2.7	Simple input capture functions	546
31.2.8	Simple one pulse functions	546
31.2.9	HRTIM configuration functions	546

31.2.10	HRTIM timer configuration and control functions	547
31.2.11	Peripheral State functions	548
31.2.12	Detailed description of functions	548
31.3	HRTIM Firmware driver defines	596
31.3.1	HRTIM	596
32	HAL HSEM Generic Driver	645
32.1	HSEM Firmware driver API description	645
32.1.1	How to use this driver	645
32.1.2	HSEM Take and Release functions	646
32.1.3	HSEM Set and Get Key functions	646
32.1.4	HSEM IRQ handler management and Notification functions	646
32.1.5	Detailed description of functions	646
32.2	HSEM Firmware driver defines	649
32.2.1	HSEM	649
33	HAL I2C Generic Driver	651
33.1	I2C Firmware driver registers structures	651
33.1.1	I2C_InitTypeDef	651
33.1.2	__I2C_HandleTypeDef	651
33.2	I2C Firmware driver API description	653
33.2.1	How to use this driver	653
33.2.2	Initialization and de-initialization functions	656
33.2.3	IO operation functions	656
33.2.4	Peripheral State, Mode and Error functions	658
33.2.5	Detailed description of functions	658
33.3	I2C Firmware driver defines	672
33.3.1	I2C	672
34	HAL I2C Extension Driver	679
34.1	I2CEx Firmware driver API description	679
34.1.1	I2C peripheral Extended features	679
34.1.2	How to use this driver	679
34.1.3	Extended features functions	679
34.1.4	Detailed description of functions	679

34.2	I2CEx Firmware driver defines	681
34.2.1	I2CEx	681
35	HAL I2S Generic Driver	683
35.1	I2S Firmware driver registers structures	683
35.1.1	I2S_InitTypeDef	683
35.1.2	__I2S_HandleTypeDef	684
35.2	I2S Firmware driver API description	685
35.2.1	How to use this driver	685
35.2.2	IO operation functions	686
35.2.3	Peripheral State and Errors functions	687
35.2.4	Initialization and de-initialization functions	687
35.2.5	Detailed description of functions	688
35.3	I2S Firmware driver defines	694
35.3.1	I2S	694
36	HAL I2S Extension Driver	700
36.1	I2SEEx Firmware driver API description	700
36.1.1	I2S Extension features	700
36.1.2	IO operation functions	700
36.1.3	Detailed description of functions	700
37	HAL IRDA Generic Driver	703
37.1	IRDA Firmware driver registers structures	703
37.1.1	IRDA_InitTypeDef	703
37.1.2	IRDA_HandleTypeDef	703
37.2	IRDA Firmware driver API description	704
37.2.1	How to use this driver	705
37.2.2	Initialization and Configuration functions	706
37.2.3	IO operation functions	706
37.2.4	Peripheral State and Error functions	709
37.2.5	Detailed description of functions	709
37.3	IRDA Firmware driver defines	718
37.3.1	IRDA	718

38	HAL IWDG Generic Driver	727
38.1	IWDG Firmware driver registers structures	727
38.1.1	IWDG_InitTypeDef	727
38.1.2	IWDG_HandleTypeDef	727
38.2	IWDG Firmware driver API description	727
38.2.1	IWDG Generic features	727
38.2.2	How to use this driver	728
38.2.3	Initialization and Start functions	728
38.2.4	IO operation functions	728
38.2.5	Detailed description of functions	728
38.3	IWDG Firmware driver defines	729
38.3.1	IWDG	729
39	HAL JPEG Generic Driver	731
39.1	JPEG Firmware driver registers structures	731
39.1.1	JPEG_ConfTypeDef	731
39.1.2	JPEG_HandleTypeDef	731
39.2	JPEG Firmware driver API description	733
39.2.1	How to use this driver	733
39.2.2	Initialization and de-initialization functions	734
39.2.3	Configuration functions	734
39.2.4	JPEG processing functions	735
39.2.5	JPEG Decode and Encode callback functions	735
39.2.6	JPEG IRQ handler management	736
39.2.7	Peripheral State and Error functions	736
39.2.8	Detailed description of functions	736
39.3	JPEG Firmware driver defines	745
39.3.1	JPEG	745
40	HAL LPTIM Generic Driver.....	750
40.1	LPTIM Firmware driver registers structures	750
40.1.1	LPTIM_ClockConfigTypeDef	750
40.1.2	LPTIM_ULPClockConfigTypeDef	750
40.1.3	LPTIM_TriggerConfigTypeDef	750

40.1.4	LPTIM_InitTypeDef	751
40.1.5	LPTIM_HandleTypeDef	751
40.2	LPTIM Firmware driver API description	752
40.2.1	How to use this driver	752
40.2.2	Initialization and de-initialization functions	753
40.2.3	LPTIM Start Stop operation functions	753
40.2.4	LPTIM Read operation functions	754
40.2.5	LPTIM IRQ handler and callbacks	754
40.2.6	Peripheral State functions	754
40.2.7	Detailed description of functions	754
40.3	LPTIM Firmware driver defines	765
40.3.1	LPTIM	765
41	HAL LTDC Generic Driver	772
41.1	LTDC Firmware driver registers structures	772
41.1.1	LTDC_ColorTypeDef	772
41.1.2	LTDC_InitTypeDef	772
41.1.3	LTDC_LayerCfgTypeDef	773
41.1.4	LTDC_HandleTypeDef	774
41.2	LTDC Firmware driver API description	775
41.2.1	How to use this driver	775
41.2.2	Initialization and Configuration functions	776
41.2.3	IO operation functions	776
41.2.4	Peripheral Control functions	776
41.2.5	Peripheral State and Errors functions	777
41.2.6	Detailed description of functions	777
41.3	LTDC Firmware driver defines	789
41.3.1	LTDC	789
42	HAL MDIOS Generic Driver	796
42.1	MDIOS Firmware driver registers structures	796
42.1.1	MDIOS_InitTypeDef	796
42.1.2	MDIOS_HandleTypeDef	796
42.2	MDIOS Firmware driver API description	796

42.2.1	How to use this driver	796
42.2.2	Initialization and Configuration functions	796
42.2.3	IO operation functions	797
42.2.4	Peripheral Control functions	797
42.2.5	Detailed description of functions	797
42.3	MDIOS Firmware driver defines	802
42.3.1	MDIOS	802
43	HAL MDMA Generic Driver	809
43.1	MDMA Firmware driver registers structures	809
43.1.1	MDMA_InitTypeDef	809
43.1.2	MDMA_LinkNodeTypeDef	810
43.1.3	MDMA_LinkNodeConfTypeDef	811
43.1.4	__MDMA_HandleTypeDef	812
43.2	MDMA Firmware driver API description	813
43.2.1	How to use this driver	813
43.2.2	Initialization and de-initialization functions	815
43.2.3	Linked list operation functions	815
43.2.4	IO operation functions	815
43.2.5	State and Errors functions	816
43.2.6	Detailed description of functions	816
43.3	MDMA Firmware driver defines	822
43.3.1	MDMA	822
44	HAL MMC Generic Driver	831
44.1	MMC Firmware driver registers structures	831
44.1.1	HAL_MMC_CardInfoTypeDef	831
44.1.2	MMC_HandleTypeDef	831
44.1.3	HAL_MMC_CardCSDTypeDef	832
44.1.4	HAL_MMC_CardCIDTypedef	835
44.2	MMC Firmware driver API description	836
44.2.1	How to use this driver	836
44.2.2	Initialization and de-initialization functions	837
44.2.3	IO operation functions	837

44.2.4	Peripheral Control functions	838
44.2.5	Detailed description of functions	838
44.3	MMC Firmware driver defines	846
44.3.1	MMC	846
45	HAL MMC Extension Driver	854
45.1	MMCEx Firmware driver API description	854
45.1.1	How to use this driver	854
45.1.2	Multibuffer functions	854
45.1.3	Detailed description of functions	854
46	HAL NAND Generic Driver	857
46.1	NAND Firmware driver registers structures	857
46.1.1	NAND_IDTypeDef.	857
46.1.2	NAND_AddressTypeDef.	857
46.1.3	NAND_DeviceConfigTypeDef.	857
46.1.4	NAND_HandleTypeDef.	858
46.2	NAND Firmware driver API description	858
46.2.1	How to use this driver	858
46.2.2	NAND Initialization and de-initialization functions	859
46.2.3	NAND Input and Output functions	859
46.2.4	NAND Control functions	859
46.2.5	NAND State functions	860
46.2.6	Detailed description of functions	860
46.3	NAND Firmware driver defines	867
46.3.1	NAND	867
47	HAL NOR Generic Driver	868
47.1	NOR Firmware driver registers structures	868
47.1.1	NOR_IDTypeDef.	868
47.1.2	NOR_CFITTypeDef.	868
47.1.3	NOR_HandleTypeDef.	868
47.2	NOR Firmware driver API description	869
47.2.1	How to use this driver	869
47.2.2	NOR Initialization and de_initialization functions	869

47.2.3	NOR Input and Output functions	870
47.2.4	NOR Control functions	870
47.2.5	NOR State functions	870
47.2.6	Detailed description of functions	870
47.3	NOR Firmware driver defines	876
47.3.1	NOR	876
48	HAL OPAMP Generic Driver	877
48.1	OPAMP Firmware driver registers structures	877
48.1.1	OPAMP_InitTypeDef	877
48.1.2	OPAMP_HandleTypeDef	878
48.2	OPAMP Firmware driver API description	878
48.2.1	OPAMP Peripheral Features	878
48.2.2	How to use this driver	879
48.2.3	Initialization and de-initialization functions	880
48.2.4	IO operation functions	880
48.2.5	Peripheral Control functions	880
48.2.6	Peripheral State functions	880
48.2.7	Detailed description of functions	881
48.3	OPAMP Firmware driver defines	884
48.3.1	OPAMP	884
49	HAL OPAMP Extension Driver	887
49.1	OPAMPEx Firmware driver API description	887
49.1.1	Extended IO operation functions	887
49.1.2	Peripheral Control functions	887
49.1.3	Detailed description of functions	887
50	HAL PCD Generic Driver	889
50.1	PCD Firmware driver registers structures	889
50.1.1	PCD_HandleTypeDef	889
50.2	PCD Firmware driver API description	890
50.2.1	How to use this driver	890
50.2.2	Initialization and de-initialization functions	890
50.2.3	IO operation functions	890

50.2.4	Peripheral Control functions	891
50.2.5	Peripheral State functions	891
50.2.6	Detailed description of functions	891
50.3	PCD Firmware driver defines	899
50.3.1	PCD	899
51	HAL PCD Extension Driver	902
51.1	PCDEx Firmware driver API description	902
51.1.1	Extended features functions	902
51.1.2	Detailed description of functions	902
52	HAL PWR Generic Driver	905
52.1	PWR Firmware driver registers structures	905
52.1.1	PWR_PVTypeDef	905
52.2	PWR Firmware driver API description	905
52.2.1	Initialization and De-Initialization functions	905
52.2.2	Peripheral Control functions	905
52.2.3	Detailed description of functions	907
52.3	PWR Firmware driver defines	912
52.3.1	PWR	912
53	HAL PWR Extension Driver	918
53.1	PWREx Firmware driver registers structures	918
53.1.1	PWREx_AVDTypeDef	918
53.1.2	PWREx_WakeupPinTypeDef	918
53.2	PWREx Firmware driver API description	918
53.2.1	Power supply control functions	918
53.2.2	Low power control functions	919
53.2.3	Peripherals control functions	921
53.2.4	Power Monitoring functions	922
53.2.5	Detailed description of functions	923
53.3	PWREx Firmware driver defines	933
53.3.1	PWREx	933
54	HAL QSPI Generic Driver	939

54.1	QSPI Firmware driver registers structures	939
54.1.1	QSPI_InitTypeDef	939
54.1.2	QSPI_HandleTypeDef	939
54.1.3	QSPI_CommandTypeDef	940
54.1.4	QSPI_AutoPollingTypeDef	941
54.1.5	QSPI_MemoryMappedTypeDef	941
54.2	QSPI Firmware driver API description	941
54.2.1	How to use this driver	941
54.2.2	Initialization and Configuration functions	943
54.2.3	IO operation functions	944
54.2.4	Peripheral Control and State functions	944
54.2.5	Detailed description of functions	945
54.3	QSPI Firmware driver defines	954
54.3.1	QSPI	954
55	HAL RCC Generic Driver	961
55.1	RCC Firmware driver registers structures	961
55.1.1	RCC_PLLInitTypeDef	961
55.1.2	RCC_OscInitTypeDef	962
55.1.3	RCC_ClkInitTypeDef	962
55.2	RCC Firmware driver API description	963
55.2.1	RCC specific features	963
55.2.2	RCC Limitations	964
55.2.3	Initialization and de-initialization functions	964
55.2.4	Peripheral Control functions	965
55.2.5	Detailed description of functions	965
55.3	RCC Firmware driver defines	969
55.3.1	RCC	970
56	HAL RCC Extension Driver	1015
56.1	RCCEx Firmware driver registers structures	1015
56.1.1	RCC_PLL2InitTypeDef	1015
56.1.2	RCC_PLL3InitTypeDef	1015
56.1.3	PLL1_ClocksTypeDef	1016

56.1.4	PLL2_ClocksTypeDef	1016
56.1.5	PLL3_ClocksTypeDef	1017
56.1.6	RCC_PeriphCLKInitTypeDef	1017
56.1.7	RCC_CRSInitTypeDef	1019
56.1.8	RCC_CRSSynchroInfoTypeDef	1020
56.2	RCCEEx Firmware driver API description.....	1021
56.2.1	Extended Peripheral Control functions	1021
56.2.2	Detailed description of functions	1021
56.3	RCCEEx Firmware driver defines	1028
56.3.1	RCCEEx	1028
57	HAL RNG Generic Driver	1076
57.1	RNG Firmware driver registers structures	1076
57.1.1	RNG_InitTypeDef	1076
57.1.2	RNG_HandleTypeDef	1076
57.2	RNG Firmware driver API description	1076
57.2.1	How to use this driver	1076
57.2.2	Initialization and de-initialization functions	1076
57.2.3	Peripheral Control functions	1077
57.2.4	Peripheral State functions	1077
57.2.5	Detailed description of functions	1077
57.3	RNG Firmware driver defines	1080
57.3.1	RNG	1080
58	HAL RTC Generic Driver.....	1084
58.1	RTC Firmware driver registers structures.....	1084
58.1.1	RTC_InitTypeDef	1084
58.1.2	RTC_TimeTypeDef	1084
58.1.3	RTC_DateTypeDef	1085
58.1.4	RTC_AlarmTypeDef	1085
58.1.5	RTC_HandleTypeDef	1086
58.2	RTC Firmware driver API description	1086
58.2.1	RTC Operating Condition	1087
58.2.2	Backup Domain Reset	1087

58.2.3	Backup Domain Access	1087
58.2.4	How to use RTC Driver	1087
58.2.5	RTC and low power modes	1087
58.2.6	Initialization and de-initialization functions	1087
58.2.7	RTC Time and Date functions	1088
58.2.8	RTC Alarm functions	1088
58.2.9	Peripheral Control functions	1088
58.2.10	Peripheral State functions	1089
58.2.11	Detailed description of functions	1089
58.3	RTC Firmware driver defines	1095
58.3.1	RTC	1095
59	HAL RTC Extension Driver	1106
59.1	RTCEx Firmware driver registers structures	1106
59.1.1	RTC_TamperTypeDef	1106
59.2	RTCEx Firmware driver API description	1106
59.2.1	How to use this driver	1107
59.2.2	RTC TimeStamp and Tamper functions	1107
59.2.3	RTC Wake-up functions	1108
59.2.4	Extended Peripheral Control functions	1108
59.2.5	Extended features functions	1109
59.2.6	Detailed description of functions	1109
59.3	RTCEx Firmware driver defines	1120
59.3.1	RTCEx	1120
60	HAL SAI Generic Driver	1138
60.1	SAI Firmware driver registers structures	1138
60.1.1	SAI_PdmInitTypeDef	1138
60.1.2	SAI_InitTypeDef	1138
60.1.3	SAI_FrameInitTypeDef	1140
60.1.4	SAI_SlotInitTypeDef	1140
60.1.5	__SAI_HandleTypeDef	1141
60.2	SAI Firmware driver API description	1142
60.2.1	How to use this driver	1142

60.2.2	Initialization and de-initialization functions	1144
60.2.3	IO operation functions.	1144
60.2.4	Peripheral State and Errors functions	1145
60.2.5	Detailed description of functions	1145
60.3	SAI Firmware driver defines	1152
60.3.1	SAI	1152
61	HAL SAI Extension Driver	1161
61.1	SAIEx Firmware driver registers structures	1161
61.1.1	SAIEx_PdmMicDelayParamTypeDef	1161
61.2	SAIEx Firmware driver API description	1161
61.2.1	Extended features functions	1161
61.2.2	Detailed description of functions	1161
62	HAL SDRAM Generic Driver	1163
62.1	SDRAM Firmware driver registers structures.	1163
62.1.1	SDRAM_HandleTypeDef	1163
62.2	SDRAM Firmware driver API description	1163
62.2.1	How to use this driver	1163
62.2.2	SDRAM Initialization and de_initialization functions	1164
62.2.3	SDRAM Input and Output functions	1164
62.2.4	SDRAM Control functions	1164
62.2.5	SDRAM State functions	1164
62.2.6	Detailed description of functions	1165
62.3	SDRAM Firmware driver defines	1171
62.3.1	SDRAM	1172
63	HAL SD Generic Driver	1173
63.1	SD Firmware driver registers structures	1173
63.1.1	HAL_SD_CardInfoTypeDef.	1173
63.1.2	SD_HandleTypeDef	1173
63.1.3	HAL_SD_CardCSDTypedef	1174
63.1.4	HAL_SD_CardCIDTypedef.	1177
63.1.5	HAL_SD_CardStatusTypedef	1178

63.2	SD Firmware driver API description	1179
63.2.1	How to use this driver	1179
63.2.2	Initialization and de-initialization functions	1180
63.2.3	IO operation functions.....	1180
63.2.4	Peripheral Control functions	1181
63.2.5	Detailed description of functions.....	1181
63.3	SD Firmware driver defines	1189
63.3.1	SD	1189
64	HAL SD Extension Driver	1197
64.1	SDEx Firmware driver API description	1197
64.1.1	How to use this driver	1197
64.1.2	Multibuffer functions	1197
64.1.3	Detailed description of functions	1197
65	HAL SMARTCARD Generic Driver	1200
65.1	SMARTCARD Firmware driver registers structures	1200
65.1.1	SMARTCARD_InitTypeDef	1200
65.1.2	SMARTCARD_AdvFeatureInitTypeDef	1201
65.1.3	SMARTCARD_HandleTypeDef	1202
65.2	SMARTCARD Firmware driver API description	1203
65.2.1	How to use this driver	1203
65.2.2	Initialization and Configuration functions	1205
65.2.3	IO operation functions.....	1205
65.2.4	Peripheral State and Errors functions	1207
65.2.5	Detailed description of functions	1208
65.3	SMARTCARD Firmware driver defines	1216
65.3.1	SMARTCARD	1216
66	HAL SMARTCARD Extension Driver	1226
66.1	SMARTCARDEX Firmware driver API description	1226
66.1.1	SMARTCARD peripheral extended features	1226
66.1.2	Peripheral Control functions	1226
66.1.3	Detailed description of functions	1226

66.2	SMARTCARDEX Firmware driver defines	1227
66.2.1	SMARTCARDEX.....	1227
67	HAL SMBUS Generic Driver	1231
67.1	SMBUS Firmware driver registers structures.....	1231
67.1.1	SMBUS_InitTypeDef.....	1231
67.1.2	SMBUS_HandleTypeDef	1232
67.2	SMBUS Firmware driver API description	1232
67.2.1	How to use this driver	1233
67.2.2	Initialization and de-initialization functions	1234
67.2.3	IO operation functions.....	1234
67.2.4	Peripheral State and Errors functions	1235
67.2.5	Detailed description of functions	1235
67.3	SMBUS Firmware driver defines	1243
67.3.1	SMBUS.....	1243
68	HAL SPDIFRX Generic Driver	1250
68.1	SPDIFRX Firmware driver registers structures	1250
68.1.1	SPDIFRX_InitTypeDef	1250
68.1.2	SPDIFRX_SetDataFormatTypeDef	1251
68.1.3	SPDIFRX_HandleTypeDef	1251
68.2	SPDIFRX Firmware driver API description	1252
68.2.1	How to use this driver	1252
68.2.2	Initialization and de-initialization functions	1254
68.2.3	IO operation functions.....	1254
68.2.4	Peripheral State and Errors functions	1255
68.2.5	Detailed description of functions	1255
68.3	SPDIFRX Firmware driver defines	1260
68.3.1	SPDIFRX	1260
69	HAL SPI Generic Driver	1266
69.1	SPI Firmware driver registers structures.....	1266
69.1.1	SPI_InitTypeDef	1266
69.1.2	__SPI_HandleTypeDef	1268

69.2	SPI Firmware driver API description	1269
69.2.1	How to use this driver	1269
69.2.2	Initialization and de-initialization functions	1270
69.2.3	IO operation functions.....	1270
69.2.4	Peripheral State and Errors functions	1271
69.2.5	Detailed description of functions.....	1271
69.3	SPI Firmware driver defines	1279
69.3.1	SPI	1279
70	HAL SPI Extension Driver	1291
70.1	SPIEx Firmware driver API description.....	1291
70.1.1	IO operation functions.....	1291
70.1.2	Detailed description of functions.....	1291
71	HAL SRAM Generic Driver.....	1293
71.1	SRAM Firmware driver registers structures	1293
71.1.1	SRAM_HandleTypeDef.....	1293
71.2	SRAM Firmware driver API description.....	1293
71.2.1	How to use this driver	1293
71.2.2	SRAM Initialization and de_initialization functions.....	1294
71.2.3	SRAM Input and Output functions.....	1294
71.2.4	SRAM Control functions	1294
71.2.5	SRAM State functions.....	1294
71.2.6	Detailed description of functions.....	1295
71.3	SRAM Firmware driver defines	1300
71.3.1	SRAM.....	1300
72	HAL SWPMI Generic Driver.....	1301
72.1	SWPMI Firmware driver registers structures	1301
72.1.1	SWPMI_InitTypeDef	1301
72.1.2	SWPMI_HandleTypeDef	1301
72.2	SWPMI Firmware driver API description.....	1302
72.2.1	How to use this driver	1302
72.2.2	Initialization and Configuration functions.....	1302

72.2.3	IO operation methods	1303
72.2.4	SWPMI IRQ handler and callbacks	1303
72.2.5	Peripheral Control methods	1304
72.2.6	Detailed description of functions	1304
72.3	SWPMI Firmware driver defines	1310
72.3.1	SWPMI	1310
73	HAL TIM Generic Driver	1315
73.1	TIM Firmware driver registers structures	1315
73.1.1	TIM_Base_InitTypeDef	1315
73.1.2	TIM_OC_InitTypeDef	1315
73.1.3	TIM_OnePulse_InitTypeDef	1316
73.1.4	TIM_IC_InitTypeDef	1317
73.1.5	TIM_Encoder_InitTypeDef	1318
73.1.6	TIM_ClockConfigTypeDef	1318
73.1.7	TIM_ClearInputConfigTypeDef	1319
73.1.8	TIM_MasterConfigTypeDef	1319
73.1.9	TIM_SlaveConfigTypeDef	1319
73.1.10	TIM_BreakDeadTimeConfigTypeDef	1320
73.1.11	TIM_HandleTypeDef	1321
73.2	TIM Firmware driver API description	1321
73.2.1	TIM Generic features	1321
73.2.2	How to use this driver	1322
73.2.3	TIM Output Compare functions	1322
73.2.4	TIM PWM functions	1323
73.2.5	TIM Input Capture functions	1323
73.2.6	TIM One Pulse functions	1324
73.2.7	IRQ handler management	1324
73.2.8	Peripheral Control functions	1324
73.2.9	TIM Callbacks functions	1325
73.2.10	Peripheral State functions	1325
73.2.11	Detailed description of functions	1325
73.3	TIM Firmware driver defines	1356

73.3.1	TIM	1356
74	HAL TIM Extension Driver	1377
74.1	TIMEx Firmware driver registers structures	1377
74.1.1	TIM_HallSensor_InitTypeDef	1377
74.1.2	TIMEx_BreakInputConfigTypeDef	1377
74.2	TIMEx Firmware driver API description	1377
74.2.1	TIM Extended features	1377
74.2.2	How to use this driver	1378
74.2.3	Peripheral Control functions	1378
74.2.4	Extended Callbacks functions	1379
74.2.5	Extended Peripheral State functions	1379
74.2.6	Detailed description of functions	1379
74.3	TIMEx Firmware driver defines	1393
74.3.1	TIMEx	1394
75	HAL UART Generic Driver	1400
75.1	UART Firmware driver registers structures	1400
75.1.1	UART_InitTypeDef	1400
75.1.2	UART_AdvFeatureInitTypeDef	1401
75.1.3	UART_HandleTypeDef	1402
75.2	UART Firmware driver API description	1403
75.2.1	How to use this driver	1403
75.2.2	Initialization and Configuration functions	1404
75.2.3	IO operation functions	1405
75.2.4	Peripheral Control functions	1405
75.2.5	Peripheral State and Error functions	1406
75.2.6	Detailed description of functions	1406
75.3	UART Firmware driver defines	1418
75.3.1	UART	1418
76	HAL UART Extension Driver	1437
76.1	UARTEEx Firmware driver registers structures	1437
76.1.1	UART_WakeUpTypeDef	1437

76.2	UARTEEx Firmware driver API description	1437
76.2.1	UART peripheral extended features	1437
76.2.2	Initialization and Configuration functions	1437
76.2.3	Peripheral Control functions	1438
76.2.4	Detailed description of functions	1438
76.3	UARTEEx Firmware driver defines	1440
76.3.1	UARTEEx	1440
77	HAL USART Generic Driver	1442
77.1	USART Firmware driver registers structures	1442
77.1.1	USART_InitTypeDef	1442
77.1.2	USART_HandleTypeDef	1443
77.2	USART Firmware driver API description	1444
77.2.1	How to use this driver	1444
77.2.2	Initialization and Configuration functions	1445
77.2.3	IO operation functions	1445
77.2.4	Peripheral State and Error functions	1447
77.2.5	Detailed description of functions	1447
77.3	USART Firmware driver defines	1456
77.3.1	USART	1456
78	HAL USART Extension Driver	1468
78.1	USARTEEx Firmware driver defines	1468
78.1.1	USARTEEx	1468
79	HAL WWDG Generic Driver	1469
79.1	WWDG Firmware driver registers structures	1469
79.1.1	WWDG_InitTypeDef	1469
79.1.2	WWDG_HandleTypeDef	1469
79.2	WWDG Firmware driver API description	1469
79.2.1	WWDG specific features	1469
79.2.2	How to use this driver	1470
79.2.3	Initialization and Configuration functions	1470
79.2.4	IO operation functions	1470

79.2.5	Detailed description of functions	1471
79.3	WWDG Firmware driver defines	1472
79.3.1	WWDG	1472
80	FAQs	1476
Revision history		1479
Contents		1480
List of tables		1512
List of figures		1513
Disclaimer		1514

List of tables

Table 1.	Acronyms and definitions	3
Table 2.	HAL driver files	5
Table 3.	User-application files	6
Table 4.	APis classification	9
Table 5.	List of devices supported by HAL drivers.	10
Table 6.	HAL API naming rules	12
Table 7.	Macros handling interrupts and specific clock configurations	13
Table 8.	Callback functions	14
Table 9.	HAL generic APIs	15
Table 10.	HAL extension APIs	16
Table 11.	Define statements used for HAL configuration	19
Table 12.	Description of GPIO_InitTypeDef structure	20
Table 13.	Description of EXTI configuration macros	22
Table 14.	MSP functions	28
Table 15.	Timeout values	31
Table 16.	Document revision history	1479

List of figures

Figure 1.	Example of project template	7
Figure 2.	Adding family-specific functions	16
Figure 3.	Adding new peripherals	17
Figure 4.	Updating existing APIs.	17
Figure 5.	File inclusion model.	18
Figure 6.	HAL driver model	25

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.

© 2018 STMicroelectronics – All rights reserved