UM1850
User manual

Description of STM32F1xx HAL drivers

Introduction

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

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF1 for STM32F1 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  - All embedded software utilities coming with a full set of examples.

The HAL drivers layer provides a generic multi instance simple 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 knowing in-depth how to use the MCU. This structure improves the library code reusability and guarantees an easy portability on 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 drivers 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), etc..

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

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

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.

Contents

List of tables

List of figures

1 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
CPU	Central Processing Unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
ETH	Ethernet controller
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HCD	USB Host Controller Driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controler
MSP	MCU Specific Package
NAND	NAND Flash memory
NOR	Nor Flash memory
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RTC	Real-time clock
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface

Acronym	Definition
SysTick	System tick timer
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

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

The HAL main features are the following:

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

2.1 HAL and user-application files

2.1.1 HAL driver files

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

Table 2: HAL drivers files

File	Description
stm32f1xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32f1xx_hal_adc.c, stm32f1xx_hal_irda.c, …*
stm32f1xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example: stm32f1xx_hal_adc.h, stm32f1xx_hal_irda.h, …*

File	Description
stm32f1xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example: stm32f1xx_hal_adc_ex.c, stm32f1xx_hal_dma_ex.c, …*
stm32f1xx_hal_ppp_ex.h	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32f1xx_hal_adc_ex.h, stm32f1xx_hal_dma_ex.h, …*
stm32f1xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
stm32f1xx_hal.h	stm32f1xx_hal.c header file
stm32f1xx_hal_msp_template.c	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32f1xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros.

2.1.2 User-application files

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

Table 3: User-application files

File	Description
system_stm32f1xx.c	This file contains SystemInit() which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows to : • relocate the vector table in internal SRAM.
startup_stm32f1xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
stm32f1xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
stm32f1xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f1xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.

File	Description
stm32f1xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32f1xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in Systick ISR. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • the call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

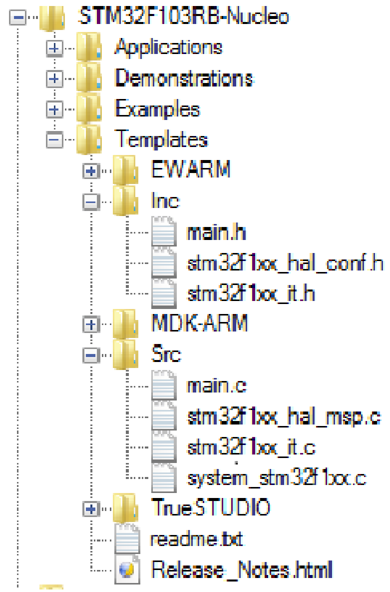
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

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

- It contains 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 to configure 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

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

Figure 1: Example of project template



2.2 HAL data structures

Each HAL driver can contain the following data structures:

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

2.2.1 Peripheral handle structures

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

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

The peripheral handle is used for the following purposes:

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

An example of peripheral structure is shown below:

```
typedef struct
{
USART_TypeDef *Instance; /* USART registers base address */
USART_InitTypeDef Init; /* Usart communication parameters */
uint8_t *pTxBuffPtr;/* Pointer to Usart Tx transfer Buffer */
uint16_t TxXferSize; /* Usart Tx Transfer size */
__IO uint16_t TxXferCount;/* Usart Tx Transfer Counter */
```

```
uint8_t *pRxBuffPtr;/* Pointer to Usart Rx transfer Buffer */
uint16_t RxXferSize; /* Usart Rx Transfer size */
__IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
HAL_LockTypeDef Lock; /* Locking object */
__IO HAL_USART_StateTypeDef State; /* Usart communication state */
__IO HAL_USART_ErrorTypeDef ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

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

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

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

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

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

2.2.2 Initialization and configuration structure

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

```
typedef struct
{
uint32_t BaudRate; /*!< This member configures the UART communication baudrate.*/
uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received
in a frame.*/
uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
uint32_t Parity; /*!< Specifies the parity mode. */
uint32_t Mode; /*!< Specifies wether the Receive or Transmit mode is enabled or
disabled.*/
uint32_t HwFlowCtl; /*!< Specifies wether the hardware flow control mode is enabled
or disabled.*/
```

```
uint32_t OverSampling; /*!< Specifies wether the Over sampling 8 is enabled or
disabled,
to achieve higher speed (up to fPCLK/8).*/
}UART_InitTypeDef;
```

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

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

2.2.3 Specific process structures

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

Example:

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

2.3 API classification

The HAL APIs are classified into three categories:

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

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

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

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

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

```
#if defined (STM32F101xG) || defined (STM32F103x6)|| defined (STM32F103xB) ||
defined (STM32F105xC) || defined (STM32F107xC) || defined (STM32F103xE) || defined
(STM32F103xG)
/* ADC multimode */HAL_StatusTypeDef
HAL_ADCEx_MultiModeStart_DMA(ADC_HandleTypeDef *hadc, uint32_t *pData,
uint32_tLength);
HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA(ADC_HandleTypeDef *hadc);
#endif /* STM32F101xG || defined STM32F103x6 || defined STM32F103xB || defined
STM32F105xC || defined STM32F107xC || defined STM32F103xE || defined STM32F103xG */
```

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

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

Table 4: APIs classification

	Generic file	Extension file
Common APIs	X	X [1]
Family specific APIs		X
Device specific APIs		X

Notes:

[1] In some cases, the implementation for a specific device part number may change . In this case the generic API is declared as weak function in the extension file. The API is implemented again to overwrite the default function

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

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

2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_hal.c stm32f1xx_hal.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_adc.c stm32f1xx_hal_adc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_adc_ex.c stm32f1xx_hal_adc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_can.c stm32f1xx_hal_can.h	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_cec.c stm32f1xx_hal_cec.h	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No
stm32f1xx_hal_cortex.c stm32f1xx_hal_cortex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_crc.c stm32f1xx_hal_crc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_dac.c stm32f1xx_hal_dac.h	Yes	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes
stm32f1xx_hal_dac_ex.c stm32f1xx_hal_dac_ex.h	Yes	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_hal_dma.c stm32f1xx_hal_dma.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_dma_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_eth.c stm32f1xx_hal_eth.h	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes
stm32f1xx_hal_flash.c stm32f1xx_hal_flash.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_flash_ex.c stm32f1xx_hal_flash_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_gpio.c stm32f1xx_hal_gpio.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_gpio_ex.c stm32f1xx_hal_gpio_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_hcd.c stm32f1xx_hal_hcd.h	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_hal_i2c.c stm32f1xx_hal_i2c.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_i2s.c stm32f1xx_hal_i2s.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes
stm32f1xx_hal_irda.c stm32f1xx_hal_irda.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_iwdg.c stm32f1xx_hal_iwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_msp_template.c	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stm32f1xx_hal_nand.c stm32f1xx_hal_nand.h	No	No	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_nor.c stm32f1xx_hal_nor.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_pccard.c stm32f1xx_hal_pccard.h	No	No	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_pcd.c stm32f1xx_hal_pcd.h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_hal_pcd_ex.c stm32f1xx_hal_pcd_ex.h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rcc.c stm32f1xx_hal_rcc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rcc_ex.c stm32f1xx_hal_rcc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rtc.c stm32f1xx_hal_rtc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_rtc_ex.c stm32f1xx_hal_rtc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_sd.c stm32f1xx_hal_sd.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_smartcard.c stm32f1xx_hal_smartcard.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_spi.c stm32f1xx_hal_spi.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_hal_sram.c stm32f1xx_hal_sram.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_hal_tim.c stm32f1xx_hal_tim.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_tim_ex.c stm32f1xx_hal_tim_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_uart.c stm32f1xx_hal_uart.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_usart.c stm32f1xx_hal_usart.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_hal_wwdg.c stm32f1xx_hal_wwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f1xx_ll_fsmc.c stm32f1xx_ll_fsmc.h	No	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No
stm32f1xx_ll_sdmmc.c stm32f1xx_ll_sdmmc.h	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No

Files	VALUE		ACCESS				USB		PERFORMANCE				OTG	Ethernet
	STM32F1 00xB	STM32F1 00xE	STM32F1 01x6	STM32F1 01xB	STM32F1 01xE	STM32F1 01xG	STM32F1 02x6	STM32F1 02xB	STM32F1 03x6	STM32F1 03xB	STM32F1 03xE	STM32F1 03xG	STM32F1 05xC	STM32F1 07xC
stm32f1xx_ll_usb.c stm32f1xx_ll_usb.h	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

2.5 HAL drivers rules

2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 7: HAL API naming rules

	Generic	Family specific	Device specific
File names	*stm32f1xx_hal_ppp (c/h)*	*stm32f1xx_hal_ppp_ex (c/h)*	*stm32f1xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function* *HAL_PPP_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32F1xx reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in stm32f1xxx.h header file. stm32f1xxx.h corresponds to stm32f100xb.h, stm32f100xe.h, stm32f101x6.h, stm32f101xb.h, stm32f101xe.h, stm32f101xg.h, stm32f102x6.h, stm32f102xb.h, stm32f103x6.h, stm32f103xb.h, stm32f103xe.h, stm32f103xg.h, stm32f105xc.h and stm32f107xc.h.
- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (e.g. ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (e.g. ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypedef (e.g DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (e.g. HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named 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

2.5.2 HAL general naming rules

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

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

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

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

Table 8: Macros handling interrupts and specific clock configurations

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

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the stm32f1xx_hal_cortex.c file.

- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example : STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)".
- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

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

- The macros defined below are used:
  - Conditional macro: #define ABS(x) (((x) > 0) ? (x) : -(x))
  - Pseudo-code macro (multiple instructions macro):

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

2.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32f1xx_it.c
- User callback functions.

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

There are three types of user callbacks functions:

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

Table 9: Callback functions

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

2.6 HAL generic APIs

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

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

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

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

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

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

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

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

Table 10: HAL generic APIs

Function Group	Common API Name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests

Function Group	Common API Name	Description
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in runtime the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in runtime the error that occurred during IT routine

2.7 HAL extension APIs

2.7.1 HAL extension model overview

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

The extension model consists of an additional file, stm32f1xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32f1xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 11: HAL extension APIs

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

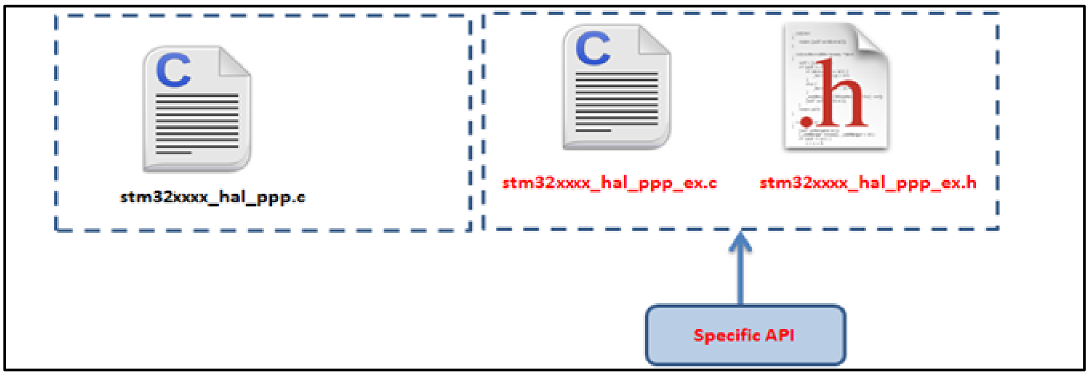
2.7.2 HAL extension model cases

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

Case1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32f1xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

Figure 2: Adding device-specific functions



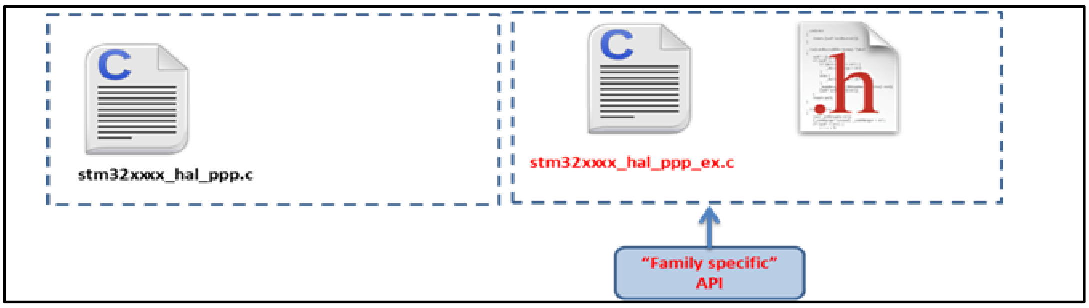
Example: stm32f1xx_hal_adc_ex.c/h

```
#if defined(STM32F101xG) || defined (STM32F103x6) || defined (STM32F103xB) ||
defined (STM32F105xC) ||
defined (STM32F107xC) || defined (STM32F103xE) || defined(STM32F103xG)
/* ADC multimode */
HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA(ADC_HandleTypeDef *hadc, uint32_t
*pData, uint32_t Length);
HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA(ADC_HandleTypeDef *hadc);
#endif /* STM32F101xG || defined STM32F103x6 || defined STM32F103xB || defined
STM32F105xC ||
defined STM32F107xC || defined STM32F103xE || defined STM32F103xG */
```

Case2: Adding a family-specific function

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

Figure 3: Adding family-specific functions

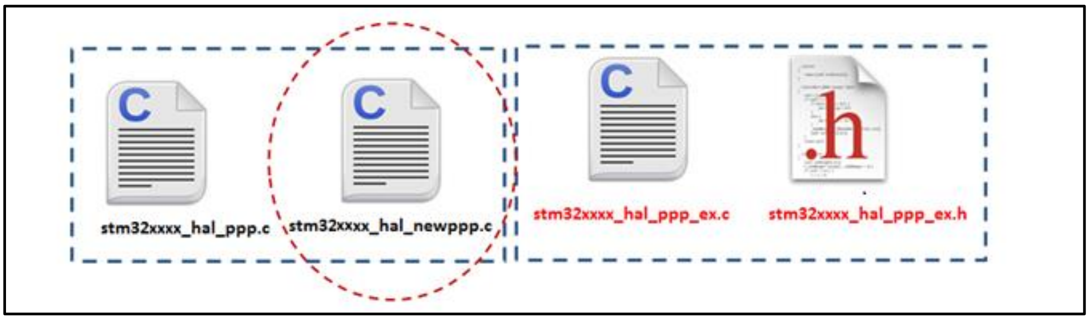


Case3 : Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in stm32f1xx_hal_newppp.c. However the inclusion of this file is selected in the stm32lxx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

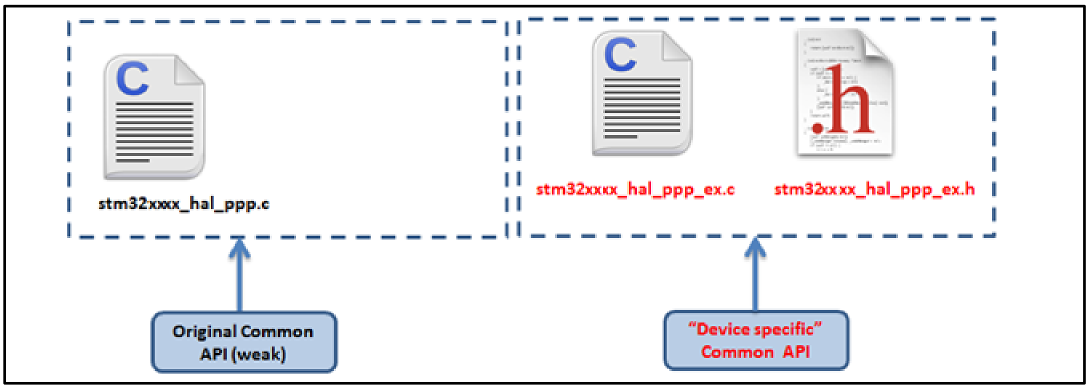


Example: stm32f1xx_hal_lcd.c/h

Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the stm32f1xx_hal_ppp_ex.c extension file, while the generic API is defined as *weak*, so that the compiler will overwrite the original routine by the new defined function.

Figure 5: Updating existing APIs



Case5 : Updating existing data structures

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

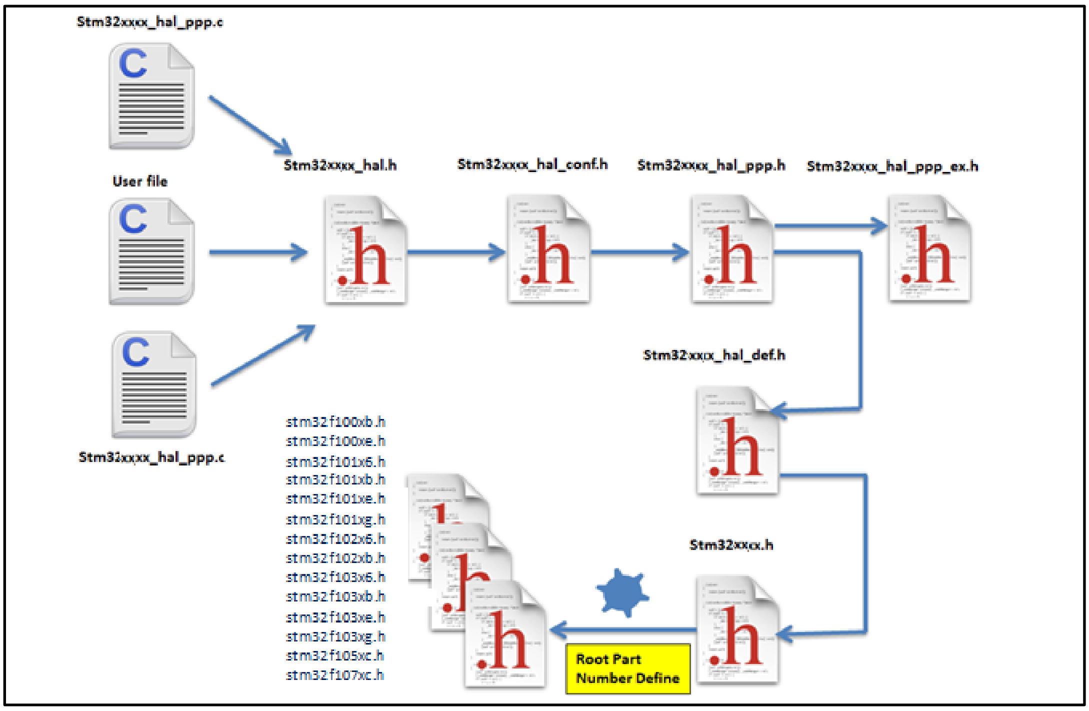
Example:

```
#if defined (STM32F100xB)
typedef struct
{
(…)
}PPP_InitTypeDef;
#endif /* STM32F100xB */
```

2.8 File inclusion model

The header of the common HAL driver file (stm32f1xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 6: File inclusion model



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

```
/*********************************************************************
* @file stm32f1xx_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
(…)
```

2.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32f1xx_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 stm32f1xx_hal_def.h file calls the stm32f1xx.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**
  - Macros defining NULL and HAL_MAX_DELAY

```
#ifndef NULL
#define NULL 0
#endif
#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)
```

2.10 HAL configuration

The configuration file, *stm32f1xx_hal_conf.h,* allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

Table 12: Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 Hz on STM3210C-EVAL, otherwise 8 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE start up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	8 000 000 Hz
LSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz
LSE_STARTUP_TIMEOUT	Timeout for LSE start up, expressed in ms	5000
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE

The stm32f1xx_hal_conf_template.h file is located in the HAL drivers *Inc* folder. It should be copied to the user folder, renamed and modified as described above.

By default, the values defined in the stm32f1xx_hal_conf_template.h file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

2.11 HAL system peripheral handling

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

2.11.1 Clock

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

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

Some peripheral clocks are not derived from the system clock (RTC, USB…). In this case, the clock configuration is performed by an extended API defined in stm32f1xx_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 retreiving various clock configurations (system clock, HCLK, PCLK1, PCLK2, …)
- MCO and CSS configuration functions

A set of macros are defined in stm32f1xx_hal_rcc.h and stm32f1xx_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.

2.11.2 GPIOs

GPIO HAL APIs are the following:

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

In addition to standard GPIO modes (input, output, analog), 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 stm32f1xx_it.c and implement HAL_GPIO_EXTI_Callback()

The table below describes the GPIO_InitTypeDef structure field.

Table 13: Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT : Input Floating – GPIO_MODE_OUTPUT_PP : Output Push Pull – GPIO_MODE_OUTPUT_OD : Output Open Drain – GPIO_MODE_AF_PP : Alternate Function Push Pull – GPIO_MODE_AF_OD : Alternate Function Open Drain – GPIO_MODE_ANALOG : Analog mode • External Interrupt Mode – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • External Event Mode – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_LOW GPIO_SPEED_MEDIUM GPIO_SPEED_HIGH

Please find below typical GPIO configuration examples:

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

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

- Configuring PA0 as external interrupt with falling edge sensitivity:

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

2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, stm32f1xx_hal_cortex.c, provides APIs to handle NVIC and Systick. The supported APIs include:

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

2.11.4 PWR

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

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

2.11.5 EXTI

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

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

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

Table 14: Description of EXTI configuration macros

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

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32f1xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback().

2.11.6 DMA

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

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

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

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

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

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enablse 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.

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

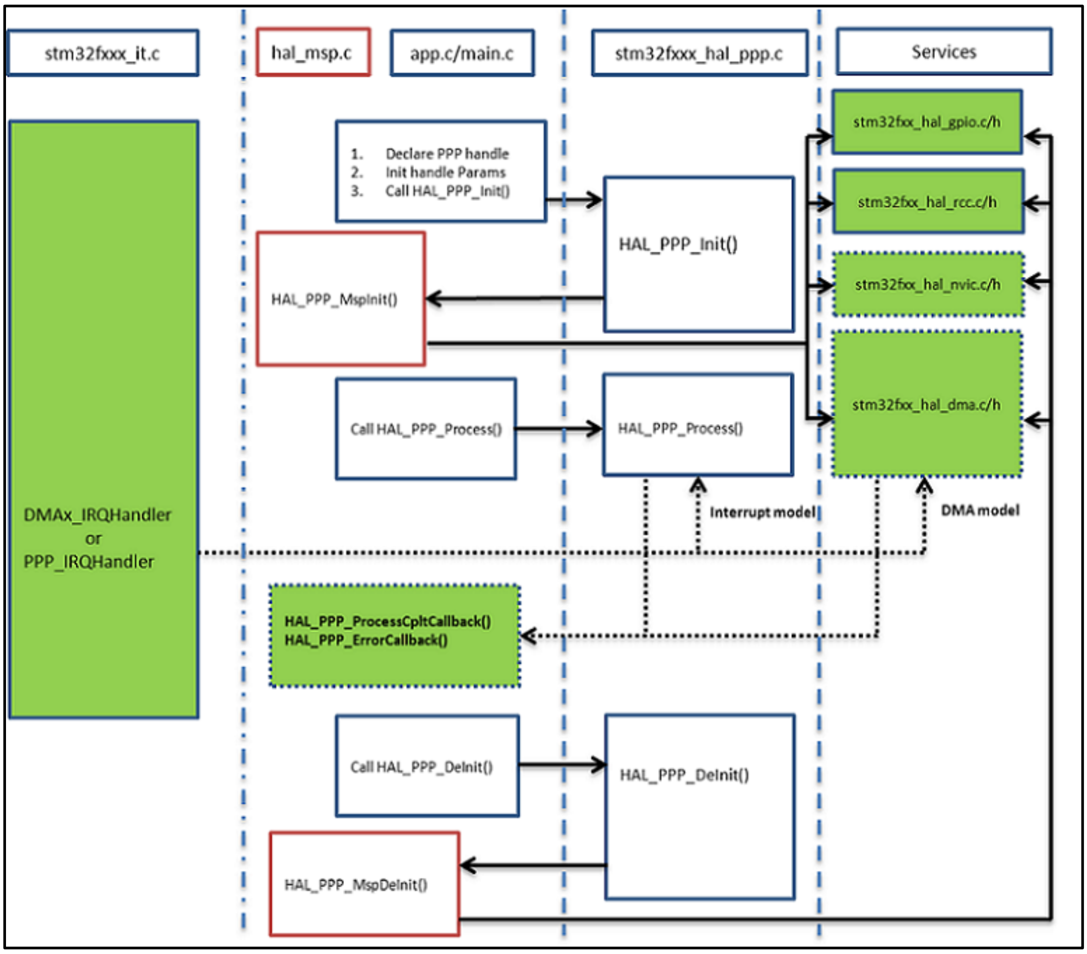
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

2.12 How to use HAL drivers

2.12.1 HAL usage models

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

Figure 7: HAL driver model



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

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

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

2.12.2 HAL initialization

2.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file stm32f1xx_hal.c.

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

2.12.2.2 System clock initialization

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

```
void SystemClock_Config(void)
{
RCC_ClkInitTypeDef clkinitstruct = {0};
RCC_OscInitTypeDef oscinitstruct = {0};

/* Configure PLLs ------------------------------------------------------*/
/* PLL2 configuration: PLL2CLK = (HSE/HSEPrediv2Value)*PLL2MUL=(25/5)*8=40 MHz */
/* PREDIV1 configuration: PREDIV1CLK = PLL2CLK / HSEPredivValue = 40 / 5 = 8 MHz */
/* PLL configuration: PLLCLK = PREDIV1CLK * PLLMUL = 8 * 9 = 72 MHz */
/* Enable HSE Oscillator and activate PLL with HSE as source */
oscinitstruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
oscinitstruct.HSEState = RCC_HSE_ON;
oscinitstruct.HSEPredivValue = RCC_HSE_PREDIV_DIV5;
oscinitstruct.Prediv1Source = RCC_PREDIV1_SOURCE_PLL2;
oscinitstruct.PLL.PLLState = RCC_PLL_ON;
oscinitstruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
oscinitstruct.PLL.PLLMUL = RCC_PLL_MUL9;
oscinitstruct.PLL2.PLL2State = RCC_PLL2_ON;
oscinitstruct.PLL2.PLL2MUL = RCC_PLL2_MUL8;
oscinitstruct.PLL2.HSEPrediv2Value = RCC_HSE_PREDIV2_DIV5;
if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)
{ /* Initialization Error */
while(1);
}

/* Select PLL as system clock source and configure the HCLK/PCLK1/PCLK2 clock
dividers */
clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK I RCC_CLOCKTYPE_HCLK I
RCC_CLOCKTYPE_PCLK1 I RCC_CLOCKTYPE_PCLK2);
clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
```

```
clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;
if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2)!= HAL_OK)
{ /* Initialization Error */
while(1);
}
}
```

2.12.2.3 HAL MSP initialization process

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

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

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

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

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

The HAL MSP callback is implemented inside the *stm32f1xx_hal_msp.c* file in the user folders. An *stm32f1xx_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.

stm32f1xx_hal_msp.c file contains the following functions:

Table 15: MSP functions

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

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

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

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

2.12.3 HAL IO operation process

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

- Polling mode
- Interrupt mode
- DMA mode

2.12.3.1 Polling mode

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

The example below shows the typical polling mode processing sequence :

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

2.12.3.2 Interrupt mode

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

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

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

To use a process in interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *stm32f1xx_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)
{
}
```

*stm32f1xx_it.c*file:

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

2.12.3.3 DMA mode

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

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

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32f1xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
PPP_TypeDef *Instance; /* Register base address */
```

```
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

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

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

An example of use is illustrated below:

main.c file:

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

stm32f1xx_it.c file:

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

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

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

2.12.4 Timeout and error management

2.12.4.1 Timeout management

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

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

The timeout possible value are the following:

Table 16: Timeout values

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

Notes:

[(1)]HAL_MAX_DELAY is defined in the stm32f1xx_hal_def.h as 0xFFFFFFFF

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

Example of fixed timeout:

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

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

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
(…)
timeout = HAL_GetTick() + Timeout;
(…)
```

```
while(ProcessOngoing)
{
(…)
if(Timeout != HAL_MAX_DELAY)
{
if(HAL_GetTick() >= timeout)
{
/* Process unlocked */
__HAL_UNLOCK(hppp);
hppp->State= HAL_PPP_STATE_TIMEOUT;
return hppp->State;
}
}
(…)
}
```

2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

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

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

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

- Timeout error: the following statement is used when a timeout error occurs: while (Process ongoing)

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

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

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

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

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
__IO HAL_PPP_StateTypeDef State; /* PPP state */
```

```
__IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
(…)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

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

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

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

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

2.12.4.3 Run-time checking

The HAL implements run-time failure detection by checking the input values of all HAL drivers functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL drivers' 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 *stm32f1xx_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 stm32f1xx_hal_conf.h:

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

```
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

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

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

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

3 HAL System Driver

3.1 HAL Firmware driver API description

The following section lists the various functions of the HAL library.

3.1.1 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

3.1.2 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 source of time base 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.
- ***HAL_Init()***
- ***HAL_DeInit()***
- ***HAL_MspInit()***
- ***HAL_MspDeInit()***
- ***HAL_InitTick()***

3.1.3 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
- ***HAL_IncTick()***
- ***HAL_GetTick()***
- ***HAL_Delay()***
- ***HAL_SuspendTick()***
- ***HAL_ResumeTick()***
- ***HAL_GetHalVersion()***
- ***HAL_GetREVID()***
- ***HAL_GetDEVID()***
- ***HAL_DBGMCU_EnableDBGSleepMode()***
- ***HAL_DBGMCU_DisableDBGSleepMode()***
- ***HAL_DBGMCU_EnableDBGStopMode()***
- ***HAL_DBGMCU_DisableDBGStopMode()***
- ***HAL_DBGMCU_EnableDBGStandbyMode()***
- ***HAL_DBGMCU_DisableDBGStandbyMode()***

3.1.4 HAL_Init

Function Name	**HAL_StatusTypeDef HAL_Init (void)**
Function Description	This function configures the Flash prefetch, Configures time base source, NVIC and Low level hardware Note: This function is called at the beginning of program after reset and before the clock configuration Note: The time base configuration is based on MSI clock when exiting from Reset.
Return values	• HAL status

3.1.5 HAL_DeInit

Function Name	**HAL_StatusTypeDef HAL_DeInit (void)**
Function Description	This function de-Initializes common part of the HAL and stops the source of time base.
Return values	• HAL status

3.1.6 HAL_MspInit

Function Name	**void HAL_MspInit (void)**
Function Description	Initializes the MSP.
Return values	• None

3.1.7 HAL_MspDeInit

Function Name	**void HAL_MspDeInit (void)**
Function Description	DeInitializes the MSP.
Return values	• None

3.1.8 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

3.1.9 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

3.1.10 HAL_GetTick

Function Name	**uint32_t HAL_GetTick (void)**
Function Description	Provides a tick value in millisecond.
Return values	• tick value

3.1.11 HAL_Delay

Function Name	**void HAL_Delay (__IO uint32_t Delay)**
Function Description	This function provides accurate delay (in milliseconds) based on variable incremented.
Parameters	• **Delay:** specifies the delay time length, in milliseconds.
Return values	• None

3.1.12 HAL_SuspendTick

Function Name	**void HAL_SuspendTick (void)**
Function Description	Suspend Tick increment.

Return values	• None

3.1.13 HAL_ResumeTick

Function Name	**void HAL_ResumeTick (void)**
Function Description	Resume Tick increment.
Return values	• None

3.1.14 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)

3.1.15 HAL_GetREVID

Function Name	**uint32_t HAL_GetREVID (void)**
Function Description	Returns the device revision identifier.
Return values	• Device revision identifier

3.1.16 HAL_GetDEVID

Function Name	**uint32_t HAL_GetDEVID (void)**
Function Description	Returns the device identifier.
Return values	• Device identifier

3.1.17 HAL_DBGMCU_EnableDBGSleepMode

Function Name	**void HAL_DBGMCU_EnableDBGSleepMode (void)**
Function Description	Enable the Debug Module during SLEEP mode.
Return values	• None

3.1.18 HAL_DBGMCU_DisableDBGSleepMode

Function Name	**void HAL_DBGMCU_DisableDBGSleepMode (void)**
Function Description	Disable the Debug Module during SLEEP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in

debug mode (not accessible by the user software in normal mode).

Return values	• None

3.1.19 HAL_DBGMCU_EnableDBGStopMode

Function Name	**void HAL_DBGMCU_EnableDBGStopMode (void)**
Function Description	Enable the Debug Module during STOP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in debug mode (not accessible by the user software in normal mode).
Return values	• None

3.1.20 HAL_DBGMCU_DisableDBGStopMode

Function Name	**void HAL_DBGMCU_DisableDBGStopMode (void)**
Function Description	Disable the Debug Module during STOP mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in debug mode (not accessible by the user software in normal mode).
Return values	• None

3.1.21 HAL_DBGMCU_EnableDBGStandbyMode

Function Name	**void HAL_DBGMCU_EnableDBGStandbyMode (void)**
Function Description	Enable the Debug Module during STANDBY mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in debug mode (not accessible by the user software in normal mode).
Return values	• None

3.1.22 HAL_DBGMCU_DisableDBGStandbyMode

Function Name	**void HAL_DBGMCU_DisableDBGStandbyMode (void)**
Function Description	Disable the Debug Module during STANDBY mode Note: On devices STM32F10xx8 and STM32F10xxB, STM32F101xC/D/E and STM32F103xC/D/E, STM32F101xF/G and STM32F103xF/G STM32F10xx4 and STM32F10xx6 Debug registers DBGMCU_IDCODE and DBGMCU_CR are accessible only in

debug mode (not accessible by the user software in normal mode).

Return values	• None

3.2 HAL Firmware driver defines

The following section lists the various define and macros of the module.

3.2.1 HAL

HAL

HAL Private Constants

__STM32F1xx_HAL_VERSION_MAIN	[31:24] main version
__STM32F1xx_HAL_VERSION_SUB1	[23:16] sub1 version
__STM32F1xx_HAL_VERSION_SUB2	[15:8] sub2 version
__STM32F1xx_HAL_VERSION_RC	[7:0] release candidate
__STM32F1xx_HAL_VERSION	
IDCODE_DEVID_MASK	

4 HAL ADC Generic Driver

4.1 ADC Firmware driver registers structures

4.1.1 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32f1xx_hal_adc.h

Data Fields

- ***uint32_t DataAlign***
- ***uint32_t ScanConvMode***
- ***uint32_t ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***uint32_t DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConv***

Field Documentation

- ***uint32_t ADC_InitTypeDef::DataAlign*** Specifies ADC data alignment to right (MSB on register bit 11 and LSB on register bit 0) (default setting) or to left (if regular group: MSB on register bit 15 and LSB on register bit 4, if injected group (MSB kept as signed value due to potential negative value after offset application): MSB on register bit 14 and LSB on register bit 3). This parameter can be a value of ***ADC_Data_align***
- ***uint32_t ADC_InitTypeDef::ScanConvMode*** Configures the sequencer of regular and injected groups. 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'/'InjectedNbrOfConversion' and each channel rank). Scan direction is upward: from rank1 to rank 'n'. This parameter can be a value of ***ADC_Scan_mode*** Note: For regular group, this parameter should be enabled in conversion either by polling (HAL_ADC_Start with Discontinuous mode and NbrOfDiscConversion=1) or by DMA (HAL_ADC_Start_DMA), but not by interruption (HAL_ADC_Start_IT): in scan mode, interruption is triggered only on the the last conversion of the sequence. All previous conversions would be overwritten by the last one. Injected group used with scan mode has not this constraint: each rank has its own result register, no data is overwritten.
- ***uint32_t ADC_InitTypeDef::ContinuousConvMode*** Specifies whether the conversion is performed in single mode (one conversion) or continuous mode for regular group, after the selected trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfConversion*** Specifies the number of ranks that will be converted within the regular group sequencer. To use 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.
- ***uint32_t ADC_InitTypeDef::DiscontinuousConvMode*** Specifies whether the conversions sequence of regular group 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 regular group (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*** Selects the external event used to trigger the conversion start of regular group. If set to ADC_SOFTWARE_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of ***ADC_External_trigger_source_Regular***

4.1.2 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32f1xx_hal_adc.h

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel*** Specifies the channel to configure into ADC regular group. This parameter can be a value of ***ADC_channels*** Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability. Note: On STM32F1 devices with several ADC: Only ADC1 can access internal measurement channels (VrefInt/TempSensor) Note: On STM32F10xx8 and STM32F10xxB devices: A low-amplitude voltage glitch may be generated (on ADC input 0) on the PA0 pin, when the ADC is converting with injection trigger. It is advised to distribute the analog channels so that Channel 0 is configured as an injected channel. Refer to errata sheet of these devices for more details.
- ***uint32_t ADC_ChannelConfTypeDef::Rank*** Specifies the rank in the regular group sequencer This parameter can be a value of ***ADC_regular_rank*** Note: In case of need 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 can be adjusted)
- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime*** Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits). This parameter can be a value of ***ADC_sampling_times*** Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/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, parameters TS_vrefint, TS_temp (values rough order: 5us to 17.1us min).

4.1.3 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32f1xx_hal_adc.h

Data Fields

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

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode*** Configures the ADC analog watchdog mode: single/all channels, regular/injected group. This parameter can be a value of ***ADC_analog_watchdog_mode***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel*** Selects which ADC channel to monitor by analog watchdog. This parameter has an effect only if watchdog mode is configured on single channel (parameter WatchdogMode) This parameter can be a value of ***ADC_channels***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::ITMode*** Specifies 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*** Configures the ADC analog watchdog High threshold value. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold*** Configures the ADC analog watchdog High threshold value. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber*** Reserved for future use, can be set to 0

4.1.4 ADC_HandleTypeDef

ADC_HandleTypeDef is defined in the stm32f1xx_hal_adc.h

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***__IO uint32_t NbrOfConversionRank***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_ADC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***ADC_TypeDef* ADC_HandleTypeDef::Instance*** Register base address
- ***ADC_InitTypeDef ADC_HandleTypeDef::Init*** ADC required parameters
- ***__IO uint32_t ADC_HandleTypeDef::NbrOfConversionRank*** ADC conversion rank counter

- ***DMA_HandleTypeDef* ADC_HandleTypeDef::DMA_Handle*** Pointer DMA Handler
- ***HAL_LockTypeDef ADC_HandleTypeDef::Lock*** ADC locking object
- ***__IO HAL_ADC_StateTypeDef ADC_HandleTypeDef::State*** ADC communication state
- ***__IO uint32_t ADC_HandleTypeDef::ErrorCode*** ADC Error code

4.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

4.2.1 ADC peripheral features

- 12-bit resolution
- Interrupt generation at the end of regular conversion, end of injected conversion, and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for automatic conversion of channel 0 to channel 'n'.
- Data alignment with in-built data coherency.
- Channel-wise programmable sampling time.
- ADC conversion Regular or Injected groups.
- External trigger (timer or EXTI) with configurable polarity for both regular and injected groups.
- DMA request generation for transfer of conversions data of regular group.
- Multimode Dual mode (available on devices with 2 ADCs or more).
- Configurable DMA data storage in Multimode Dual mode (available on devices with 2 DCs or more).
- Configurable delay between conversions in Dual interleaved mode (available on devices with 2 DCs or more).
- ADC calibration
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

4.2.2 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.
   - One clock setting is mandatory: ADC clock (core and conversion clock):
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
     - PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIVx ;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);

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, groups regular/injected, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ..., of regular group) 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 injected group parameters (conversion trigger, sequencer, ..., of injected group) and the channels for injected group parameters (channel number, channel rank into sequencer, ..., into injected group) using function HAL_ADCEx_InjectedConfigChannel().
4. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().
5. Optionally, for devices with several ADC instances: configure the multimode parameters using function HAL_ADCEx_MultiModeConfigChannel().

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() (or for injected group: HAL_ADCEx_InjectedPollForConversion())
     - Retrieve conversion results using function HAL_ADC_GetValue() (or for injected group: HAL_ADCEx_InjectedGetValue())
     - 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) (or for injected group: HAL_ADCEx_InjectedConvCpltCallback())
  - Retrieve conversion results using function HAL_ADC_GetValue() (or for injected group: HAL_ADCEx_InjectedGetValue())
  - 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()
- For devices with several ADCs: ADC multimode conversion with transfer by DMA:
  - Activate the ADC peripheral (slave) and start conversions using function HAL_ADC_Start()
  - Activate the ADC peripheral (master) and start conversions using function HAL_ADCEx_MultiModeStart_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 (master) using function HAL_ADCEx_MultiModeStop_DMA()
  - Stop conversion and disable the ADC peripheral (slave) using function HAL_ADC_Stop_IT()

Callback functions must be implemented in user program:

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

Deinitialization of ADC

1. Disable the ADC interface
   - ADC clock can be hard reset and disabled at RCC top level.
   - Hard reset of ADC peripherals using macro __ADCx_FORCE_RESET(), __ADCx_RELEASE_RESET().
   - ADC clock disable using the equivalent macro/functions as configuration step.
     - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC
     - PeriphClkInit.AdcClockSelection = RCC_ADCPLLCLK2_OFF

   – 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)

4.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- De-initialize the ADC.
- ***HAL_ADC_Init()***
- ***HAL_ADC_DeInit()***
- ***HAL_ADC_MspInit()***
- ***HAL_ADC_MspDeInit()***

4.2.4 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.
- ***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()***

4.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog
- ***HAL_ADC_ConfigChannel()***
- ***HAL_ADC_AnalogWDGConfig()***

4.2.6 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
- ***HAL_ADC_GetState()***
- ***HAL_ADC_GetError()***

4.2.7 HAL_ADC_Init

Function Name	**HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)**
Function Description	Initializes 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 (clock source APB2). See commented example code below that can be copied and uncommented into HAL_ADC_MspInit(). • Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef". • This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".

4.2.8 HAL_ADC_DeInit

Function Name	**HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)**

Function Description	Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• HAL status

4.2.9 HAL_ADC_MspInit

Function Name	**void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)**
Function Description	Initializes the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• None

4.2.10 HAL_ADC_MspDeInit

Function Name	**void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)**
Function Description	DeInitializes the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• None

4.2.11 HAL_ADC_Start

Function Name	**HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)**
Function Description	Enables ADC, starts conversion of regular group.
Parameters	• **hadc:** ADC handle
Return values	• HAL status

4.2.12 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.

4.2.13 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

4.2.14 HAL_ADC_PollForEvent

Function Name	**HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)**
Function Description	Poll for conversion event.
Parameters	• **hadc:** ADC handle • **EventType:** the ADC event type. This parameter can be one of the following values: ADC_AWD_EVENT: ADC Analog watchdog event. • **Timeout:** Timeout value in millisecond.
Return values	• HAL status

4.2.15 HAL_ADC_Start_IT

Function Name	**HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)**
Function Description	Enables ADC, starts conversion of regular group with interruption.

4.2.16 HAL_ADC_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)**
Function Description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interrution of end-of-conversion, disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• None

4.2.17 HAL_ADC_Start_DMA

Function Name	**HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t**

	Length)
Function Description	Enables ADC, starts conversion of regular group and transfers result through DMA.

4.2.18 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. • For devices with several ADCs: This function is for single-ADC mode only. For multimode, use the dedicated MultimodeStop function. • On STM32F1 devices, only ADC1 and ADC3 (ADC availability depending on devices) have DMA capability.

4.2.19 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	• Converted value
Notes	• Reading DR register automatically clears EOC (end of conversion of regular group) flag.

4.2.20 HAL_ADC_IRQHandler

Function Name	**void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)**
Function Description	Handles ADC interrupt request.
Parameters	• **hadc:** ADC handle
Return values	• None

4.2.21 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

4.2.22 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

4.2.23 HAL_ADC_LevelOutOfWindowCallback

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

4.2.24 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

4.2.25 HAL_ADC_ConfigChannel

Function Name	**HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)**
Function Description	Configures the the selected channel to be linked to the regular group.
Parameters	• **hadc:** ADC handle • **sConfig:** Structure of ADC channel for regular group.
Return values	• HAL status

Notes	•	In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be be disabled using function HAL_ADC_DeInit().
	•	Possibility to update parameters on the fly: This function initializes channel into regular group, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without reseting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_ChannelConfTypeDef".

4.2.26 HAL_ADC_AnalogWDGConfig

Function Name	**HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)**
Function Description	Configures the analog watchdog.
Parameters	• **hadc:** ADC handle • **AnalogWDGConfig:** Structure of ADC analog watchdog configuration
Return values	• HAL status

4.2.27 HAL_ADC_GetState

Function Name	**HAL_ADC_StateTypeDef HAL_ADC_GetState (ADC_HandleTypeDef * hadc)**
Function Description	return the ADC state
Parameters	• **hadc:** ADC handle
Return values	• HAL state

4.2.28 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

4.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

4.3.1 ADC

ADC

ADC analog watchdog mode

ADC_ANALOGWATCHDOG_NONE

ADC_ANALOGWATCHDOG_SINGLE_REG

ADC_ANALOGWATCHDOG_SINGLE_INJEC

ADC_ANALOGWATCHDOG_SINGLE_REGINJEC

ADC_ANALOGWATCHDOG_ALL_REG

ADC_ANALOGWATCHDOG_ALL_INJEC

ADC_ANALOGWATCHDOG_ALL_REGINJEC

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_TEMPSENSOR

ADC_CHANNEL_VREFINT

ADC conversion cycles

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_1CYCLE5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_7CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_13CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_28CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_41CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_55CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_71CYCLES5

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_239CYCLES5

ADC conversion group

ADC_REGULAR_GROUP

ADC_INJECTED_GROUP

ADC_REGULAR_INJECTED_GROUP

ADC data alignment

ADC_DATAALIGN_RIGHT

ADC_DATAALIGN_LEFT

ADC Error Code

HAL_ADC_ERROR_NONE	No error
HAL_ADC_ERROR_INTERNAL	ADC IP internal error: if problem of clocking, enable/disable, erroneous state
HAL_ADC_ERROR_OVR	Overrun error
HAL_ADC_ERROR_DMA	DMA transfer error

ADC Event type

ADC_AWD_EVENT	ADC Analog watchdog event
ADC_AWD1_EVENT	ADC Analog watchdog 1 event: Alternate naming for compatibility with other STM32 devices having several analog watchdogs

ADC Exported Macros

__HAL_ADC_ENABLE

Description:

- Enable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None:

__HAL_ADC_DISABLE

Description:

- Disable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None:

__HAL_ADC_ENABLE_IT

Description:

- Enable the ADC end of conversion interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be any combination of the

following values:
  - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
  - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
  - ADC_IT_AWD: ADC Analog watchdog interrupt source

Return value:

- None:

__HAL_ADC_DISABLE_IT

Description:

- Disable the ADC end of conversion interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be any combination of the following values:
  - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
  - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
  - ADC_IT_AWD: ADC Analog watchdog interrupt source

Return value:

- None:

__HAL_ADC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be any combination of the following values:
  - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
  - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
  - ADC_IT_AWD: ADC Analog watchdog interrupt source

Return value:

- None:

__HAL_ADC_GET_FLAG

Description:

- Get the selected ADC's flag status.

Parameters:

- __HANDLE__: ADC handle

- __FLAG__: ADC flag This parameter can be any combination of the following values:
  - ADC_FLAG_STRT: ADC Regular group start flag
  - ADC_FLAG_JSTRT: ADC Injected group start flag
  - ADC_FLAG_EOC: ADC End of Regular conversion flag
  - ADC_FLAG_JEOC: ADC End of Injected conversion flag
  - ADC_FLAG_AWD: ADC Analog watchdog flag

Return value:

- None:

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the ADC's pending flags.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be any combination of the following values:
  - ADC_FLAG_STRT: ADC Regular group start flag
  - ADC_FLAG_JSTRT: ADC Injected group start flag
  - ADC_FLAG_EOC: ADC End of Regular conversion flag
  - ADC_FLAG_JEOC: ADC End of Injected conversion flag
  - ADC_FLAG_AWD: ADC Analog watchdog flag

Return value:

- None:

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None:

ADC external trigger enable for regular group

ADC_EXTERNALTRIGCONVEDGE_NONE

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC External trigger selection for regular group

ADC_EXTERNALTRIGCONV_T1_CC1 < List of external triggers with generic trigger name, independently of

ADC_EXTERNALTRIGCONV_T1_CC2	
ADC_EXTERNALTRIGCONV_T2_CC2	
ADC_EXTERNALTRIGCONV_T3_TRGO	
ADC_EXTERNALTRIGCONV_T4_CC4	
ADC_EXTERNALTRIGCONV_EXT_IT11	External triggers of regular group for ADC3 only
ADC_EXTERNALTRIGCONV_T2_CC3	
ADC_EXTERNALTRIGCONV_T3_CC1	
ADC_EXTERNALTRIGCONV_T5_CC1	
ADC_EXTERNALTRIGCONV_T5_CC3	
ADC_EXTERNALTRIGCONV_T8_CC1	
ADC_EXTERNALTRIGCONV_T1_CC3	< External triggers of regular group for all ADC instances Note: TIM8_TRGO is available on ADC1 and ADC2 only in high-density and
ADC_EXTERNALTRIGCONV_T8_TRGO	
ADC_SOFTWARE_START	

ADC flags definition

ADC_FLAG_STRT	ADC Regular group start flag
ADC_FLAG_JSTRT	ADC Injected group start flag
ADC_FLAG_EOC	ADC End of Regular conversion flag
ADC_FLAG_JEOC	ADC End of Injected conversion flag
ADC_FLAG_AWD	ADC Analog watchdog flag

ADC interrupts definition

ADC_IT_EOC	ADC End of Regular Conversion interrupt source
ADC_IT_JEOC	ADC End of Injected Conversion interrupt source
ADC_IT_AWD	ADC Analog watchdog interrupt source

ADC Private Constants

ADC_ENABLE_TIMEOUT

ADC_DISABLE_TIMEOUT

ADC_STAB_DELAY_US

ADC_TEMPSENSOR_DELAY_US

ADC_FLAG_POSTCONV_ALL

ADC Private Macros

ADC_IS_ENABLE

Description:

- Verification of ADC state: enabled or disabled.

Parameters:

- __HANDLE__: ADC handle

Return value:

- SET: (ADC enabled) or RESET (ADC disabled)

ADC_IS_SOFTWARE_START_REGULAR

Description:

- Test if conversion trigger of regular group is software start or external trigger.

Parameters:

- __HANDLE__: ADC handle

Return value:

- SET: (software start) or RESET (external trigger)

ADC_IS_SOFTWARE_START_INJECTED

Description:

- Test if conversion trigger of injected group is software start or external trigger.

Parameters:

- __HANDLE__: ADC handle

Return value:

- SET: (software start) or RESET (external trigger)

ADC_CLEAR_ERRORCODE

Description:

- Clear ADC error code (set it to error code: "no error")

Parameters:

- __HANDLE__: ADC handle

Return value:

- None:

ADC_SQR1_L_SHIFT

Description:

- Set ADC number of conversions into regular channel sequence length.

Parameters:

- _NbrOfConversion_: Regular channel sequence length

Return value:

- None:

ADC_SMPR1

Description:

- Set the ADC's sample time for channel numbers between 10 and 18.

Parameters:

- _SAMPLETIME_: Sample time parameter.
- _CHANNELNB_: Channel number.

Return value:

- None:

ADC_SMPR2

Description:

- Set the ADC's sample time for channel numbers between 0 and 9.

Parameters:

- _SAMPLETIME_: Sample time parameter.
- _CHANNELNB_: Channel number.

Return value:

- None:

ADC_SQR3_RK

Description:

- Set the selected regular channel rank for rank between 1 and 6.

Parameters:

- _CHANNELNB_: Channel number.
- _RANKNB_: Rank number.

Return value:

- None:

ADC_SQR2_RK

Description:

- Set the selected regular channel rank for rank between 7 and 12.

Parameters:

- _CHANNELNB_: Channel number.
- _RANKNB_: Rank number.

Return value:

- None:

ADC_SQR1_RK

Description:

- Set the selected regular channel rank for rank between 13 and 16.

Parameters:

- _CHANNELNB_: Channel number.
- _RANKNB_: Rank number.

Return value:

- None:

ADC_JSQR_JL_SHIFT

Description:

- Set the injected sequence length.

Parameters:

- _JSQR_JL_: Sequence length.

Return value:

- None:

ADC_JSQR_RK_JL

Description:

- Set the selected injected channel rank Note: on STM32F1 devices, channel rank position in JSQR register is depending on total number of ranks selected into injected sequencer (ranks sequence starting from 4-JL)

Parameters:

- _CHANNELNB_: Channel number.
- _RANKNB_: Rank number.
- _JSQR_JL_: Sequence length.

Return value:

- None:

ADC_CR2_CONTINUOUS

Description:

- Enable ADC continuous conversion mode.

Parameters:

- _CONTINUOUS_MODE_: Continuous mode.

Return value:

- None:

ADC_CR1_DISCONTINUOUS_NUM

Description:

- Configures the number of discontinuous conversions for the regular group channels.

Parameters:

- _NBR_DISCONTINUOUS_CONV_: Number of discontinuous conversions.

Return value:

- None:

ADC_CR1_SCAN_SET

Description:

- Enable ADC scan mode to convert multiple ranks with sequencer.

Parameters:

- _SCAN_MODE_: Scan conversion mode.

Return value:

- None:

ADC_CONVCYCLES_MAX_RANGE

Description:

- Get the maximum ADC conversion cycles on all channels.

Parameters:

- __HANDLE__: ADC handle

Return value:

- ADC: conversion cycles on all channels

IS_ADC_DATA_ALIGN

IS_ADC_SCAN_MODE

IS_ADC_EXTTRIG_EDGE

IS_ADC_CHANNEL

IS_ADC_SAMPLE_TIME

IS_ADC_REGULAR_RANK

IS_ADC_ANALOG_WATCHDOG_MODE

IS_ADC_CONVERSION_GROUP

IS_ADC_EVENT_TYPE

ADC range verification

IS_ADC_RANGE

ADC regular discontinuous mode number verification

IS_ADC_REGULAR_DISCONT_NUMBER

ADC regular nb conv verification

IS_ADC_REGULAR_NB_CONV

ADC rank into regular group

ADC_REGULAR_RANK_1

ADC_REGULAR_RANK_2

ADC_REGULAR_RANK_3

ADC_REGULAR_RANK_4

ADC_REGULAR_RANK_5

ADC_REGULAR_RANK_6

ADC_REGULAR_RANK_7

ADC_REGULAR_RANK_8

ADC_REGULAR_RANK_9

ADC_REGULAR_RANK_10

ADC_REGULAR_RANK_11

ADC_REGULAR_RANK_12

ADC_REGULAR_RANK_13

ADC_REGULAR_RANK_14

ADC_REGULAR_RANK_15

ADC_REGULAR_RANK_16

ADC sampling times

ADC_SAMPLETIME_1CYCLE_5	Sampling time 1.5 ADC clock cycle
ADC_SAMPLETIME_7CYCLES_5	Sampling time 7.5 ADC clock cycles
ADC_SAMPLETIME_13CYCLES_5	Sampling time 13.5 ADC clock cycles
ADC_SAMPLETIME_28CYCLES_5	Sampling time 28.5 ADC clock cycles
ADC_SAMPLETIME_41CYCLES_5	Sampling time 41.5 ADC clock cycles
ADC_SAMPLETIME_55CYCLES_5	Sampling time 55.5 ADC clock cycles
ADC_SAMPLETIME_71CYCLES_5	Sampling time 71.5 ADC clock cycles
ADC_SAMPLETIME_239CYCLES_5	Sampling time 239.5 ADC clock cycles

ADC sampling times all channels

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT2

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT2

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT1

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT1

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT0

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT0

ADC_SAMPLETIME_1CYCLE5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_7CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_13CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_28CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_41CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_55CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_71CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_239CYCLES5_SMPR2ALLCHANNELS

ADC_SAMPLETIME_1CYCLE5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_7CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_13CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_28CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_41CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_55CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_71CYCLES5_SMPR1ALLCHANNELS

ADC_SAMPLETIME_239CYCLES5_SMPR1ALLCHANNELS

ADC scan mode

ADC_SCAN_DISABLE

ADC_SCAN_ENABLE

5 HAL ADC Extension Driver

5.1 ADCEx Firmware driver registers structures

5.1.1 ADC_InjectionConfTypeDef

ADC_InjectionConfTypeDef is defined in the stm32f1xx_hal_adc_ex.h

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedOffset***
- ***uint32_t InjectedNbrOfConversion***
- ***uint32_t InjectedDiscontinuousConvMode***
- ***uint32_t AutoInjectedConv***
- ***uint32_t ExternalTrigInjecConv***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel*** Selection of ADC channel to configure This parameter can be a value of ***ADC_channels*** Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability. Note: On STM32F1 devices with several ADC: Only ADC1 can access internal measurement channels (VrefInt/TempSensor) Note: On STM32F10xx8 and STM32F10xxB devices: A low-amplitude voltage glitch may be generated (on ADC input 0) on the PA0 pin, when the ADC is converting with injection trigger. It is advised to distribute the analog channels so that Channel 0 is configured as an injected channel. Refer to errata sheet of these devices for more details.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank*** Rank in the injected group sequencer This parameter must be a value of ***ADCEx_injected_rank*** Note: In case of need 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 can be adjusted)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime*** Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits). This parameter can be a value of ***ADC_sampling_times*** Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/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, parameters TS_vrefint, TS_temp (values rough order: 5us to 17.1us min).
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset*** Defines the offset to be subtracted from the raw converted data (for channels set on injected group only). Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.

- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion*** Specifies the number of ranks that will be converted within the injected group sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode*** Specifies whether the conversions sequence of injected group 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. Note: For injected group, number of discontinuous ranks increment is fixed to one-by-one. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::AutoInjectedConv*** Enables or disables the selected ADC automatic injected group 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_SOFTWARE_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::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. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of ***ADCEx_External_trigger_source_Injected*** 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 behaviour in case of another parameter update on the fly) 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.

5.1.2 ADC_MultiModeTypeDef

ADC_MultiModeTypeDef is defined in the stm32f1xx_hal_adc_ex.h

Data Fields

- ***uint32_t Mode***

Field Documentation

- ***uint32_t ADC_MultiModeTypeDef::Mode*** Configures the ADC to operate in independent or multi mode. This parameter can be a value of ***ADCEx_Common_mode*** Note: In dual mode, a change of channel configuration generates a restart that can produce a loss of synchronization. It is recommended to disable dual mode before any configuration change. Note: In case of simultaneous mode used: Exactly the same sampling time should be configured for the 2 channels that will be sampled simultaneously by ACD1 and ADC2. Note: In case of interleaved mode used: To avoid overlap between conversions, maximum sampling time allowed is 7 ADC clock cycles for fast interleaved mode and 14 ADC clock cycles for slow interleaved mode. Note: Some multimode parameters are fixed on STM32F1 and can be configured on other STM32 devices with several ADC (multimode configuration structure can have additional parameters). The equivalences are:
  - Parameter 'DMAAccessMode': On STM32F1, this parameter is fixed to 1 DMA channel (one DMA channel for both ADC, DMA of ADC master). On other STM32 devices with several ADC, this is equivalent to parameter 'ADC_DMAACCESSMODE_12_10_BITS'.
  - Parameter 'TwoSamplingDelay': On STM32F1, this parameter is fixed to 7 or 14 ADC clock cycles depending on fast or slow interleaved mode selected. On other STM32 devices with several ADC, this is equivalent to parameter 'ADC_TWOSAMPLINGDELAY_7CYCLES' (for fast interleaved mode).

5.2 ADCEx Firmware driver API description

The following section lists the various functions of the ADCEx library.

5.2.1 IO operation functions

This section provides functions allowing to:

- 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.
- Start multimode and enable DMA transfer.
- Stop multimode and disable ADC DMA transfer.
- Get result of multimode conversion.
- 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.
- ***HAL_ADCEx_Calibration_Start()***
- ***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_InjectedGetValue()***
- ***HAL_ADCEx_MultiModeGetValue()***
- ***HAL_ADCEx_InjectedConvCpltCallback()***

5.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group
- Configure multimode
- *HAL_ADCEx_InjectedConfigChannel()*
- *HAL_ADCEx_MultiModeConfigChannel()*

5.2.3 HAL_ADCEx_Calibration_Start

Function Name	**HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc)**
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
Return values	• HAL status

5.2.4 HAL_ADCEx_InjectedStart

Function Name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)**
Function Description	Enables ADC, starts conversion of injected group.
Parameters	• **hadc:** ADC handle
Return values	• HAL status

5.2.5 HAL_ADCEx_InjectedStop

Function Name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)**
Function Description	Stop conversion of injected channels.
Parameters	• **hadc:** ADC handle
Return values	• None
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. • In case of auto-injection mode, HAL_ADC_Stop must be used.

5.2.6 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

5.2.7 HAL_ADCEx_InjectedStart_IT

Function Name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)**
Function Description	Enables ADC, starts conversion of injected group with interruption.

5.2.8 HAL_ADCEx_InjectedStop_IT

Function Name	**HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)**
Function Description	Stop conversion of injected channels, disable interruption of end-of-conversion.
Parameters	• **hadc:** ADC handle
Return values	• None
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.

5.2.9 HAL_ADCEx_MultiModeStart_DMA

Function Name	**HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)**
Function Description	Enables ADC, starts conversion of regular group and transfers result through DMA.

5.2.10 HAL_ADCEx_MultiModeStop_DMA

Function Name	**HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (ADC_HandleTypeDef * hadc)**
Function Description	Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.
Parameters	• **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values	•	None
Notes	•	Multimode is kept enabled after this function. To disable multimode (set with HAL_ADCEx_MultiModeConfigChannel(), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_ReInit().
	•	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.

5.2.11 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: Injected Channel1 selected ADC_INJECTED_RANK_2: Injected Channel2 selected ADC_INJECTED_RANK_3: Injected Channel3 selected ADC_INJECTED_RANK_4: Injected Channel4 selected
Return values	• None

5.2.12 HAL_ADCEx_MultiModeGetValue

Function Name	**uint32_t HAL_ADCEx_MultiModeGetValue (ADC_HandleTypeDef * hadc)**
Function Description	Returns the last ADC Master&Slave regular conversions results data in the selected multi mode.
Parameters	• **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)
Return values	• The converted data value.

5.2.13 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

5.2.14 HAL_ADCEx_InjectedConfigChannel

Function Name	**HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)**
Function Description	Configures the ADC injected group and the selected channel to be linked to the injected group.
Parameters	• **hadc:** ADC handle • **sConfigInjected:** Structure of ADC injected group and ADC channel for injected group.
Return values	• None
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 reseting the ADC. The setting of these parameters is conditioned to ADC state: this function must be called when ADC is not under conversion.

5.2.15 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:** 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 reseting the ADCs (both ADCs of the common group). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_MultiModeTypeDef". • To change back configuration from multimode to single mode, ADC must be reset (using function HAL_ADC_Init()).

5.3 ADCEx Firmware driver defines

The following section lists the various define and macros of the module.

5.3.1 ADCEx

ADCEx

ADC Extended Dual ADC Mode

ADC_MODE_INDEPENDENT	ADC dual mode disabled (ADC independent mode)
ADC_DUALMODE_REGSIMULT_INJECSIMU	ADC dual mode enabled: Combined

LT	regular simultaneous + injected simultaneous mode
ADC_DUALMODE_REGSIMULT_ALTERTRIG	ADC dual mode enabled: Combined regular simultaneous + alternate trigger mode
ADC_DUALMODE_INJECSIMULT_INTERLFAST	ADC dual mode enabled: Combined injected simultaneous + fast interleaved mode (delay between ADC sampling phases: 7 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INJECSIMULT_INTERLSLOW	ADC dual mode enabled: Combined injected simultaneous + slow Interleaved mode (delay between ADC sampling phases: 14 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INJECSIMULT	ADC dual mode enabled: Injected simultaneous mode only
ADC_DUALMODE_REGSIMULT	ADC dual mode enabled: Regular simultaneous mode only
ADC_DUALMODE_INTERLFAST	ADC dual mode enabled: Fast interleaved mode only (delay between ADC sampling phases: 7 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_INTERLSLOW	ADC dual mode enabled: Slow interleaved mode only (delay between ADC sampling phases: 14 ADC clock cycles (equivalent to parameter "TwoSamplingDelay" set to "ADC_TWOSAMPLINGDELAY_7CYCLES" on other STM32 devices))
ADC_DUALMODE_ALTERTRIG	ADC dual mode enabled: Alternate trigger mode only

ADCEx external trigger enable for injected group

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE

ADC_EXTERNALTRIGINJECCONV_EDGE_RISING

ADCEx External trigger selection for injected group

ADC_EXTERNALTRIGINJECCONV_T2_TRGO	< List of external triggers with generic trigger name, independently of
ADC_EXTERNALTRIGINJECCONV_T2_CC1	
ADC_EXTERNALTRIGINJECCONV_T3_CC4	

ADC_EXTERNALTRIGINJECCONV_T4_TRGO	
ADC_EXTERNALTRIGINJECCONV_EXT_IT15	External triggers of injected group for ADC3 only
ADC_EXTERNALTRIGINJECCONV_T4_CC3	
ADC_EXTERNALTRIGINJECCONV_T8_CC2	
ADC_EXTERNALTRIGINJECCONV_T5_TRGO	
ADC_EXTERNALTRIGINJECCONV_T5_CC4	
ADC_EXTERNALTRIGINJECCONV_T1_CC4	< External triggers of injected group for all ADC instances
ADC_EXTERNALTRIGINJECCONV_T1_TRGO	Note: TIM8_CC4 is available on ADC1 and ADC2 only in high-density and
ADC_EXTERNALTRIGINJECCONV_T8_CC4	
ADC_INJECTED_SOFTWARE_START	

ADCEx injected nb conv verification

IS_ADC_INJECTED_NB_CONV

ADCEx rank into injected group

ADC_INJECTED_RANK_1

ADC_INJECTED_RANK_2

ADC_INJECTED_RANK_3

ADC_INJECTED_RANK_4

ADC Extended Internal HAL driver trigger selection for injected group

ADC1_2_EXTERNALTRIGINJEC_T2_TRGO

ADC1_2_EXTERNALTRIGINJEC_T2_CC1

ADC1_2_EXTERNALTRIGINJEC_T3_CC4

ADC1_2_EXTERNALTRIGINJEC_T4_TRGO

ADC1_2_EXTERNALTRIGINJEC_EXT_IT15

ADC1_2_EXTERNALTRIGINJEC_T8_CC4

ADC3_EXTERNALTRIGINJEC_T4_CC3

ADC3_EXTERNALTRIGINJEC_T8_CC2

ADC3_EXTERNALTRIGINJEC_T8_CC4

ADC3_EXTERNALTRIGINJEC_T5_TRGO

ADC3_EXTERNALTRIGINJEC_T5_CC4

ADC1_2_3_EXTERNALTRIGINJEC_T1_TRGO

ADC1_2_3_EXTERNALTRIGINJEC_T1_CC4

ADC1_2_3_JSWSTART

ADC Extended Internal HAL driver trigger selection for regular group

ADC1_2_EXTERNALTRIG_T1_CC1

ADC1_2_EXTERNALTRIG_T1_CC2

ADC1_2_EXTERNALTRIG_T2_CC2

ADC1_2_EXTERNALTRIG_T3_TRGO

ADC1_2_EXTERNALTRIG_T4_CC4

ADC1_2_EXTERNALTRIG_EXT_IT11

ADC1_2_EXTERNALTRIG_T8_TRGO

ADC3_EXTERNALTRIG_T3_CC1

ADC3_EXTERNALTRIG_T2_CC3

ADC3_EXTERNALTRIG_T8_CC1

ADC3_EXTERNALTRIG_T8_TRGO

ADC3_EXTERNALTRIG_T5_CC1

ADC3_EXTERNALTRIG_T5_CC3

ADC1_2_3_EXTERNALTRIG_T1_CC3

ADC1_2_3_SWSTART

ADCEx Private Constants

ADC_PRECALIBRATION_DELAY_ADCCLOCKCYCLES

ADC_CALIBRATION_TIMEOUT

ADC_TEMPSENSOR_DELAY_US

ADCEx Private Macro

ADC_CFGR_EXTSEL

Description:

- For devices with 3 ADCs: Defines the external trigger source for regular group according to ADC into common group ADC1&ADC2 or ADC3 (some triggers with same source have different value to be programmed into ADC EXTSEL bits of CR2 register).

Parameters:

- __HANDLE__: ADC handle
- __EXT_TRIG_CONV__: External trigger selected for regular group.

Return value:

- External: trigger to be programmed into EXTSEL bits of CR2 register

ADC_CFGR_JEXTSEL

Description:

- For devices with 3 ADCs: Defines the external trigger source for injected group

	according to ADC into common group ADC1&ADC2 or ADC3 (some triggers with same source have different value to be programmed into ADC JEXTSEL bits of CR2 register). **Parameters:** • __HANDLE__: ADC handle • __EXT_TRIG_INJECTCONV__: External trigger selected for injected group. **Return value:** • External: trigger to be programmed into JEXTSEL bits of CR2 register
ADC_MULTIMODE_IS_ENABLE	**Description:** • Verification if multimode is enabled for the selected ADC (multimode ADC master or ADC slave) (applicable for devices with several ADCs) **Parameters:** • __HANDLE__: ADC handle **Return value:** • Multimode: state: RESET if multimode is disabled, other value if multimode is enabled
ADC_NONMULTIMODE_OR_MULTIMODEMASTER	**Description:** • Verification of condition for ADC start conversion: ADC must be in non-multimode, or multimode with handle of ADC master (applicable for devices with several ADCs) **Parameters:** • __HANDLE__: ADC handle **Return value:** • None:
ADC_COMMON_ADC_OTHER	**Description:** • Set handle of the other ADC sharing the common multimode settings. **Parameters:** • __HANDLE__: ADC handle

- __HANDLE_OTHER_ADC__: other ADC handle

Return value:

- None:

ADC_MULTI_SLAVE

Description:

- Set handle of the ADC slave associated to the ADC master On STM32F1 devices, ADC slave is always ADC2 (this can be different on other STM32 devices)

Parameters:

- __HANDLE_MASTER__: ADC master handle
- __HANDLE_SLAVE__: ADC slave handle

Return value:

- None:

IS_ADC_INJECTED_RANK

IS_ADC_EXTTRIGINJEC_EDGE

IS_ADC_EXTTRIG

IS_ADC_EXTTRIGINJEC

IS_ADC_MODE

6 HAL CAN Generic Driver

6.1 CAN Firmware driver registers structures

6.1.1 CAN_InitTypeDef

CAN_InitTypeDef is defined in the stm32f1xx_hal_can.h

Data Fields

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

Field Documentation

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

6.1.2 CanTxMsgTypeDef

CanTxMsgTypeDef is defined in the stm32f1xx_hal_can.h

Data Fields

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

Field Documentation

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

6.1.3 CanRxMsgTypeDef

CanRxMsgTypeDef is defined in the stm32f1xx_hal_can.h

Data Fields

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

Field Documentation

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

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

6.1.4 CAN_HandleTypeDef

CAN_HandleTypeDef is defined in the stm32f1xx_hal_can.h

Data Fields

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

Field Documentation

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

6.2 CAN Firmware driver API description

The following section lists the various functions of the CAN library.

6.2.1 How to use this driver

1. Enable the CAN controller interface clock using __HAL_RCC_CAN1_CLK_ENABLE() for CAN1 and __HAL_RCC_CAN2_CLK_ENABLE() for CAN2 In case you are using CAN2 only, you have to enable the CAN1 clock.
2. CAN pins configuration
   - Enable the clock for the CAN GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
   - Connect and configure the involved CAN pins using the following function HAL_GPIO_Init();

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

Polling mode IO operation

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

Interrupt mode IO operation

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

CAN HAL driver macros list

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

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

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

6.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the CAN.
- De-initialize the CAN.
- ***HAL_CAN_Init()***
- ***HAL_CAN_ConfigFilter()***
- ***HAL_CAN_DeInit()***
- ***HAL_CAN_MspInit()***
- ***HAL_CAN_MspDeInit()***

6.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a CAN frame message.
- Receive a CAN frame message.
- Enter CAN peripheral in sleep mode.
- Wake up the CAN peripheral from sleep mode.
- ***HAL_CAN_Transmit()***
- ***HAL_CAN_Transmit_IT()***
- ***HAL_CAN_Receive()***
- ***HAL_CAN_Receive_IT()***
- ***HAL_CAN_Sleep()***
- ***HAL_CAN_WakeUp()***
- ***HAL_CAN_IRQHandler()***
- ***HAL_CAN_TxCpltCallback()***
- ***HAL_CAN_RxCpltCallback()***
- ***HAL_CAN_ErrorCallback()***

6.2.4 Peripheral State and Error functions

This subsection provides functions allowing to :

- Check the CAN state.
- Check CAN Errors detected during interrupt process
- ***HAL_CAN_GetState()***
- ***HAL_CAN_GetError()***

6.2.5 HAL_CAN_Init

Function Name	**HAL_StatusTypeDef HAL_CAN_Init (CAN_HandleTypeDef * hcan)**
Function Description	Initializes the CAN peripheral according to the specified parameters in the CAN_InitStruct.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• HAL status

6.2.6 HAL_CAN_ConfigFilter

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

		that contains the filter configuration information.
Return values	•	None

6.2.7 HAL_CAN_DeInit

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

6.2.8 HAL_CAN_MspInit

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

6.2.9 HAL_CAN_MspDeInit

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

6.2.10 HAL_CAN_Transmit

Function Name	**HAL_StatusTypeDef HAL_CAN_Transmit (CAN_HandleTypeDef * hcan, uint32_t Timeout)**
Function Description	Initiates and transmits a CAN frame message.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **Timeout:** Specify Timeout value
Return values	• HAL status

6.2.11 HAL_CAN_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_CAN_Transmit_IT (CAN_HandleTypeDef * hcan)**
Function Description	Initiates and transmits a CAN frame message.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• HAL status

6.2.12 HAL_CAN_Receive

Function Name	**HAL_StatusTypeDef HAL_CAN_Receive (CAN_HandleTypeDef * hcan, uint8_t FIFONumber, uint32_t Timeout)**
Function Description	Receives a correct CAN frame.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **FIFONumber:** FIFO Number value • **Timeout:** Specify Timeout value
Return values	• HAL status • None

6.2.13 HAL_CAN_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_CAN_Receive_IT (CAN_HandleTypeDef * hcan, uint8_t FIFONumber)**
Function Description	Receives a correct CAN frame.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. • **FIFONumber:** Specify the FIFO number
Return values	• HAL status • None

6.2.14 HAL_CAN_Sleep

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

6.2.15 HAL_CAN_WakeUp

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

6.2.16 HAL_CAN_IRQHandler

Function Name	**void HAL_CAN_IRQHandler (CAN_HandleTypeDef * hcan)**
Function Description	Handles CAN interrupt request.
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• None

6.2.17 HAL_CAN_TxCpltCallback

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

6.2.18 HAL_CAN_RxCpltCallback

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

6.2.19 HAL_CAN_ErrorCallback

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

6.2.20 HAL_CAN_GetState

Function Name	**HAL_CAN_StateTypeDef HAL_CAN_GetState (CAN_HandleTypeDef * hcan)**
Function Description	return the CAN state
Parameters	• **hcan:** pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	• HAL state

6.2.21 HAL_CAN_GetError

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

6.3 CAN Firmware driver defines

The following section lists the various define and macros of the module.

6.3.1 CAN

CAN

CAN Error Code

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

CAN Exported Macros

__HAL_CAN_RESET_HANDLE_STATE	**Description:** • Reset CAN handle state. **Parameters:** • __HANDLE__: CAN handle.

Return value:

- None:

__HAL_CAN_ENABLE_IT

Description:

- Enable the specified CAN interrupts.

Parameters:

- __HANDLE__: CAN handle.
- __INTERRUPT__: CAN Interrupt. This parameter can be one of the following values:
  - CAN_IT_TME: Transmit mailbox empty interrupt enable
  - CAN_IT_FMP0: FIFO 0 message pending interrupt
  - CAN_IT_FF0 : FIFO 0 full interrupt
  - CAN_IT_FOV0: FIFO 0 overrun interrupt
  - CAN_IT_FMP1: FIFO 1 message pending interrupt
  - CAN_IT_FF1 : FIFO 1 full interrupt
  - CAN_IT_FOV1: FIFO 1 overrun interrupt
  - CAN_IT_WKU : Wake-up interrupt
  - CAN_IT_SLK : Sleep acknowledge interrupt
  - CAN_IT_EWG : Error warning interrupt
  - CAN_IT_EPV : Error passive interrupt
  - CAN_IT_BOF : Bus-off interrupt
  - CAN_IT_LEC : Last error code interrupt
  - CAN_IT_ERR : Error Interrupt

Return value:

- None.:

__HAL_CAN_DISABLE_IT

Description:

- Disable the specified CAN interrupts.

Parameters:

- __HANDLE__: CAN handle.
- __INTERRUPT__: CAN Interrupt. This parameter can be one of the following values:
  - CAN_IT_TME: Transmit mailbox empty interrupt enable
  - CAN_IT_FMP0: FIFO 0 message pending interrupt
  - CAN_IT_FF0 : FIFO 0 full interrupt
  - CAN_IT_FOV0: FIFO 0 overrun interrupt
  - CAN_IT_FMP1: FIFO 1 message

pending interrupt
- CAN_IT_FF1 : FIFO 1 full interrupt
- CAN_IT_FOV1: FIFO 1 overrun interrupt
- CAN_IT_WKU : Wake-up interrupt
- CAN_IT_SLK : Sleep acknowledge interrupt
- CAN_IT_EWG : Error warning interrupt
- CAN_IT_EPV : Error passive interrupt
- CAN_IT_BOF : Bus-off interrupt
- CAN_IT_LEC : Last error code interrupt
- CAN_IT_ERR : Error Interrupt

Return value:

- None.:

__HAL_CAN_MSG_PENDING

Description:

- Return the number of pending received messages.

Parameters:

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

Return value:

- The: number of pending message.

__HAL_CAN_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - CAN_TSR_RQCP0: Request MailBox0 Flag
  - CAN_TSR_RQCP1: Request MailBox1 Flag
  - CAN_TSR_RQCP2: Request MailBox2 Flag
  - CAN_FLAG_TXOK0: Transmission OK MailBox0 Flag
  - CAN_FLAG_TXOK1: Transmission OK MailBox1 Flag
  - CAN_FLAG_TXOK2: Transmission OK MailBox2 Flag
  - CAN_FLAG_TME0: Transmit mailbox 0 empty Flag
  - CAN_FLAG_TME1: Transmit mailbox

1 empty Flag
- CAN_FLAG_TME2: Transmit mailbox 2 empty Flag
- CAN_FLAG_FMP0: FIFO 0 Message Pending Flag
- CAN_FLAG_FF0: FIFO 0 Full Flag
- CAN_FLAG_FOV0: FIFO 0 Overrun Flag
- CAN_FLAG_FMP1: FIFO 1 Message Pending Flag
- CAN_FLAG_FF1: FIFO 1 Full Flag
- CAN_FLAG_FOV1: FIFO 1 Overrun Flag
- CAN_FLAG_WKU: Wake up Flag
- CAN_FLAG_SLAK: Sleep acknowledge Flag
- CAN_FLAG_SLAKI: Sleep acknowledge Flag
- CAN_FLAG_EWG: Error Warning Flag
- CAN_FLAG_EPV: Error Passive Flag
- CAN_FLAG_BOF: Bus-Off Flag

Return value:

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

__HAL_CAN_CLEAR_FLAG

Description:

- Clear the specified CAN pending flag.

Parameters:

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

Pending Flag
- CAN_FLAG_FF0: FIFO 0 Full Flag
- CAN_FLAG_FOV0: FIFO 0 Overrun Flag
- CAN_FLAG_FMP1: FIFO 1 Message Pending Flag
- CAN_FLAG_FF1: FIFO 1 Full Flag
- CAN_FLAG_FOV1: FIFO 1 Overrun Flag
- CAN_FLAG_WKU: Wake up Flag
- CAN_FLAG_SLAKI: Sleep acknowledge Flag

Return value:

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

__HAL_CAN_GET_IT_SOURCE

Description:

- Check if the specified CAN interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the CAN Handle.
- __INTERRUPT__: specifies the CAN interrupt source to check. This parameter can be one of the following values:
  - CAN_IT_TME: Transmit mailbox empty interrupt enable
  - CAN_IT_FMP0: FIFO 0 message pending interrupt
  - CAN_IT_FF0 : FIFO 0 full interrupt
  - CAN_IT_FOV0: FIFO 0 overrun interrupt
  - CAN_IT_FMP1: FIFO 1 message pending interrupt
  - CAN_IT_FF1 : FIFO 1 full interrupt
  - CAN_IT_FOV1: FIFO 1 overrun interrupt
  - CAN_IT_WKU : Wake-up interrupt
  - CAN_IT_SLK : Sleep acknowledge interrupt
  - CAN_IT_EWG : Error warning interrupt
  - CAN_IT_EPV : Error passive interrupt
  - CAN_IT_BOF : Bus-off interrupt
  - CAN_IT_LEC : Last error code interrupt
  - CAN_IT_ERR : Error Interrupt

Return value:

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

__HAL_CAN_TRANSMIT_STATUS

Description:

- Check the transmission status of a CAN Frame.

Parameters:

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

Return value:

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

__HAL_CAN_FIFO_RELEASE

Description:

- Release the specified receive FIFO.

Parameters:

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

Return value:

- None.:

__HAL_CAN_CANCEL_TRANSMIT

Description:

- Cancel a transmit request.

Parameters:

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

Return value:

- None.:

__HAL_CAN_DBG_FREEZE

Description:

- Enable or disables the DBG Freeze for CAN.

Parameters:

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

Return value:

- None:

CAN Filter FIFO

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

CAN Filter Mode

CAN_FILTERMODE_IDMASK	Identifier mask mode
CAN_FILTERMODE_IDLIST	Identifier list mode

CAN Filter Scale

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

CAN Flags

CAN_FLAG_RQCP0	Request MailBox0 flag
CAN_FLAG_RQCP1	Request MailBox1 flag
CAN_FLAG_RQCP2	Request MailBox2 flag
CAN_FLAG_TXOK0	Transmission OK MailBox0 flag
CAN_FLAG_TXOK1	Transmission OK MailBox1 flag
CAN_FLAG_TXOK2	Transmission OK MailBox2 flag
CAN_FLAG_TME0	Transmit mailbox 0 empty flag
CAN_FLAG_TME1	Transmit mailbox 0 empty flag
CAN_FLAG_TME2	Transmit mailbox 0 empty flag
CAN_FLAG_FF0	FIFO 0 Full flag
CAN_FLAG_FOV0	FIFO 0 Overrun flag
CAN_FLAG_FF1	FIFO 1 Full flag
CAN_FLAG_FOV1	FIFO 1 Overrun flag
CAN_FLAG_WKU	Wake up flag
CAN_FLAG_SLAK	Sleep acknowledge flag
CAN_FLAG_SLAKI	Sleep acknowledge flag
CAN_FLAG_EWG	Error warning flag
CAN_FLAG_EPV	Error passive flag
CAN_FLAG_BOF	Bus-Off flag

CAN Identifier Type

CAN_ID_STD	Standard Id
CAN_ID_EXT	Extended Id

CAN initialization Status

CAN_INITSTATUS_FAILED	CAN initialization failed
CAN_INITSTATUS_SUCCESS	CAN initialization OK

CAN Interrupts

CAN_IT_TME	Transmit mailbox empty interrupt
CAN_IT_FMP0	FIFO 0 message pending interrupt
CAN_IT_FF0	FIFO 0 full interrupt

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

CAN Operating Mode

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

CAN Private Constants

CAN_TIMEOUT_VALUE

CAN_TI0R_STID_BIT_POSITION

CAN_TI0R_EXID_BIT_POSITION

CAN_TDL0R_DATA0_BIT_POSITION

CAN_TDL0R_DATA1_BIT_POSITION

CAN_TDL0R_DATA2_BIT_POSITION

CAN_TDL0R_DATA3_BIT_POSITION

TSR_REGISTER_INDEX

RF0R_REGISTER_INDEX

RF1R_REGISTER_INDEX

MSR_REGISTER_INDEX

ESR_REGISTER_INDEX

CAN_TSR_RQCP0_BIT_POSITION

CAN_TSR_RQCP1_BIT_POSITION

CAN_TSR_RQCP2_BIT_POSITION

CAN_TSR_TXOK0_BIT_POSITION

CAN_TSR_TXOK1_BIT_POSITION

CAN_TSR_TXOK2_BIT_POSITION

CAN_TSR_TME0_BIT_POSITION

CAN_TSR_TME1_BIT_POSITION

CAN_TSR_TME2_BIT_POSITION

CAN_RF0R_FF0_BIT_POSITION

CAN_RF0R_FOV0_BIT_POSITION

CAN_RF1R_FF1_BIT_POSITION

CAN_RF1R_FOV1_BIT_POSITION

CAN_MSR_WKU_BIT_POSITION

CAN_MSR_SLAK_BIT_POSITION

CAN_MSR_SLAKI_BIT_POSITION

CAN_ESR_EWG_BIT_POSITION

CAN_ESR_EPV_BIT_POSITION

CAN_ESR_BOF_BIT_POSITION

CAN_FLAG_MASK

CAN_TXMAILBOX_0

CAN_TXMAILBOX_1

CAN_TXMAILBOX_2

CAN Private Macros

IS_CAN_MODE

IS_CAN_SJW

IS_CAN_BS1

IS_CAN_BS2

IS_CAN_FILTER_MODE

IS_CAN_FILTER_SCALE

IS_CAN_FILTER_FIFO

IS_CAN_IDTYPE

IS_CAN_RTR

IS_CAN_FIFO

IS_CAN_BANKNUMBER

IS_CAN_TRANSMITMAILBOX

IS_CAN_STDID

IS_CAN_EXTID

IS_CAN_DLC

IS_CAN_PRESCALER

CAN Receive FIFO Number

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

CAN Remote Transmission Request

CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame

CAN Synchronization Jump Width

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

CAN Time Quantum in Bit Segment 1

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

CAN Time Quantum in Bit Segment 2

CAN_BS2_1TQ	1 time quantum
CAN_BS2_2TQ	2 time quantum
CAN_BS2_3TQ	3 time quantum
CAN_BS2_4TQ	4 time quantum
CAN_BS2_5TQ	5 time quantum
CAN_BS2_6TQ	6 time quantum
CAN_BS2_7TQ	7 time quantum
CAN_BS2_8TQ	8 time quantum

CAN Transmit Constants

CAN_TXSTATUS_NOMAILBOX	CAN cell did not provide CAN_TxStatus_NoMailBox

7 HAL CAN Extension Driver

7.1 CANEx Firmware driver registers structures

7.1.1 CAN_FilterConfTypeDef

CAN_FilterConfTypeDef is defined in the stm32f1xx_hal_can_ex.h

Data Fields

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

Field Documentation

- ***uint32_t CAN_FilterConfTypeDef::FilterIdHigh*** Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t CAN_FilterConfTypeDef::FilterIdLow*** Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t CAN_FilterConfTypeDef::FilterMaskIdHigh*** Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t CAN_FilterConfTypeDef::FilterMaskIdLow*** Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t CAN_FilterConfTypeDef::FilterFIFOAssignment*** Specifies the FIFO (0 or 1) which will be assigned to the filter. This parameter can be a value of ***CAN_filter_FIFO***
- ***uint32_t CAN_FilterConfTypeDef::FilterNumber*** Specifies the filter which will be initialized. This parameter must be a number between Min_Data = 0 and Max_Data = 13.
- ***uint32_t CAN_FilterConfTypeDef::FilterMode*** Specifies the filter mode to be initialized. This parameter can be a value of ***CAN_filter_mode***
- ***uint32_t CAN_FilterConfTypeDef::FilterScale*** Specifies the filter scale. This parameter can be a value of ***CAN_filter_scale***
- ***uint32_t CAN_FilterConfTypeDef::FilterActivation*** Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t CAN_FilterConfTypeDef::BankNumber*** Select the start slave bank filter This parameter must be a number between Min_Data = 0 and Max_Data = 28.

7.2 CANEx Firmware driver defines

The following section lists the various define and macros of the module.

7.2.1 CANEx

CANEx

CAN Extended Private Macros

IS_CAN_FILTER_NUMBER

8 HAL CEC Generic Driver

8.1 CEC Firmware driver registers structures

8.1.1 CEC_InitTypeDef

CEC_InitTypeDef is defined in the stm32f1xx_hal_cec.h

Data Fields

- ***uint32_t TimingErrorFree***
- ***uint32_t PeriodErrorFree***
- ***uint8_t InitiatorAddress***

Field Documentation

- ***uint32_t CEC_InitTypeDef::TimingErrorFree*** Configures the CEC Bit Timing Error Mode. This parameter can be a value of ***CEC_BitTimingErrorMode***
- ***uint32_t CEC_InitTypeDef::PeriodErrorFree*** Configures the CEC Bit Period Error Mode. This parameter can be a value of ***CEC_BitPeriodErrorMode***
- ***uint8_t CEC_InitTypeDef::InitiatorAddress*** Initiator address (source logical address, sent in each header) This parameter can be a value <= 0xF

8.1.2 CEC_HandleTypeDef

CEC_HandleTypeDef is defined in the stm32f1xx_hal_cec.h

Data Fields

- ***CEC_TypeDef * Instance***
- ***CEC_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint32_t ErrorCode***
- ***HAL_LockTypeDef Lock***
- ***HAL_CEC_StateTypeDef State***

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
- ***uint8_t* CEC_HandleTypeDef::pRxBuffPtr*** Pointer to CEC Rx transfer Buffer
- ***uint16_t CEC_HandleTypeDef::RxXferSize*** CEC Rx Transfer size, 0: header received only
- ***uint32_t CEC_HandleTypeDef::ErrorCode*** For errors handling purposes, copy of ESR register in case error is reported
- ***HAL_LockTypeDef CEC_HandleTypeDef::Lock*** Locking object

- ***HAL_CEC_StateTypeDef CEC_HandleTypeDef::State*** CEC communication state

8.2 CEC Firmware driver API description

The following section lists the various functions of the CEC library.

8.2.1 How to use this driver

The CEC HAL driver can be used as follows:

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. Enable the clock for the CEC GPIOs.
   c. Configure these CEC pins as alternate function pull-up.
   d. NVIC configuration if you need to use interrupt process (HAL_CEC_Transmit_IT() and HAL_CEC_Receive_IT() APIs):
   e. Configure the CEC interrupt priority.
   f. Enable the NVIC CEC IRQ handle.
   g. The CEC interrupt is activated/deactivated by the HAL driver
3. Program the Bit Timing Error Mode and the Bit Period Error Mode in the hcec Init structure.
4. Initialize the CEC registers by calling the HAL_CEC_Init() API.
5. This API (HAL_CEC_Init()) configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_CEC_MspInit() API.

8.2.2 Initialization and Configuration functions

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

- The following parameters need to be configured:
  - TimingErrorFree
  - PeriodErrorFree
  - InitiatorAddress
- ***HAL_CEC_Init()***
- ***HAL_CEC_DeInit()***
- ***HAL_CEC_MspInit()***
- ***HAL_CEC_MspDeInit()***

8.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the CEC data transfers.

1. There are two modes of transfer:
   a. 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.
   b. No-Blocking mode: The communication is performed using Interrupts. These API's return the HAL status. The end of the data processing will be indicated through the dedicated CEC IRQ when using Interrupt mode. The HAL_CEC_TxCpltCallback(), HAL_CEC_RxCpltCallback() user callbacks will be executed respectivelly at the end of the Transmit or Receive process. The

HAL_CEC_ErrorCallback()user callback will be executed when a communication error is detected

2. Blocking mode API's are :
   a. HAL_CEC_Transmit()
   b. HAL_CEC_Receive()
3. Non-Blocking mode API's with Interrupt are :
   a. HAL_CEC_Transmit_IT()
   b. HAL_CEC_Receive_IT()
   c. HAL_CEC_IRQHandler()
4. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
   a. HAL_CEC_TxCpltCallback()
   b. HAL_CEC_RxCpltCallback()
   c. HAL_CEC_ErrorCallback()

- ***HAL_CEC_Transmit()***
- ***HAL_CEC_Receive()***
- ***HAL_CEC_Transmit_IT()***
- ***HAL_CEC_Receive_IT()***
- ***HAL_CEC_GetReceivedFrameSize()***
- ***HAL_CEC_IRQHandler()***
- ***HAL_CEC_TxCpltCallback()***
- ***HAL_CEC_RxCpltCallback()***
- ***HAL_CEC_ErrorCallback()***

8.2.4 Peripheral Control functions

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 get the error code of a failed transmission or reception.
- ***HAL_CEC_GetState()***
- ***HAL_CEC_GetError()***

8.2.5 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

8.2.6 HAL_CEC_DeInit

Function Name	**HAL_StatusTypeDef HAL_CEC_DeInit (CEC_HandleTypeDef * hcec)**
Function Description	DeInitializes the CEC peripheral.

Parameters	• **hcec:** CEC handle
Return values	• HAL status

8.2.7 HAL_CEC_MspInit

Function Name	**void HAL_CEC_MspInit (CEC_HandleTypeDef * hcec)**
Function Description	CEC MSP Init.
Parameters	• **hcec:** CEC handle
Return values	• None

8.2.8 HAL_CEC_MspDeInit

Function Name	**void HAL_CEC_MspDeInit (CEC_HandleTypeDef * hcec)**
Function Description	CEC MSP DeInit.
Parameters	• **hcec:** CEC handle
Return values	• None

8.2.9 HAL_CEC_Transmit

Function Name	**HAL_StatusTypeDef HAL_CEC_Transmit (CEC_HandleTypeDef * hcec, uint8_t DestinationAddress, uint8_t * pData, uint32_t Size, uint32_t Timeout)**
Function Description	Send data in blocking mode.
Parameters	• **hcec:** CEC handle • **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). • **Timeout:** Timeout duration.
Return values	• HAL status

8.2.10 HAL_CEC_Receive

Function Name	**HAL_StatusTypeDef HAL_CEC_Receive (CEC_HandleTypeDef * hcec, uint8_t * pData, uint32_t Timeout)**
Function Description	Receive data in blocking mode.
Parameters	• **hcec:** CEC handle • **pData:** pointer to received data buffer. • **Timeout:** Timeout duration.

Return values	•	HAL status
Notes	•	The received data size is not known beforehand, the latter is known when the reception is complete and is stored in hcec->RxXferSize. hcec->RxXferSize is the sum of opcodes + operands (0 to 14 operands max). If only a header is received, hcec->RxXferSize = 0

8.2.11 HAL_CEC_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_CEC_Transmit_IT (CEC_HandleTypeDef * hcec, uint8_t DestinationAddress, uint8_t * pData, uint32_t Size)**
Function Description	Send data in interrupt mode.
Parameters	• **hcec:** CEC handle • **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

8.2.12 HAL_CEC_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_CEC_Receive_IT (CEC_HandleTypeDef * hcec, uint8_t * pData)**
Function Description	Receive data in interrupt mode.
Parameters	• **hcec:** CEC handle • **pData:** pointer to received data buffer.
Return values	• HAL status
Notes	• The received data size is not known beforehand, the latter is known when the reception is complete and is stored in hcec->RxXferSize. hcec->RxXferSize is the sum of opcodes + operands (0 to 14 operands max). If only a header is received, hcec->RxXferSize = 0

8.2.13 HAL_CEC_GetReceivedFrameSize

Function Name	**uint32_t HAL_CEC_GetReceivedFrameSize (CEC_HandleTypeDef * hcec)**
Function Description	Get size of the received frame.
Parameters	• **hcec:** CEC handle
Return values	• Frame size

8.2.14 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

8.2.15 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

8.2.16 HAL_CEC_RxCpltCallback

Function Name	**void HAL_CEC_RxCpltCallback (CEC_HandleTypeDef * hcec)**
Function Description	Rx Transfer completed callback.
Parameters	• **hcec:** CEC handle
Return values	• None

8.2.17 HAL_CEC_ErrorCallback

Function Name	**void HAL_CEC_ErrorCallback (CEC_HandleTypeDef * hcec)**
Function Description	CEC error callbacks.
Parameters	• **hcec:** CEC handle
Return values	• None

8.2.18 HAL_CEC_GetState

Function Name	**HAL_CEC_StateTypeDef HAL_CEC_GetState (CEC_HandleTypeDef * hcec)**
Function Description	return the CEC state
Parameters	• **hcec:** CEC handle
Return values	• HAL state

8.2.19 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

8.3 CEC Firmware driver defines

The following section lists the various define and macros of the module.

8.3.1 CEC

CEC

Bit Period Error Mode

CEC_BIT_PERIOD_ERROR_MODE_STANDARD	Bit period error Standard Mode
CEC_BIT_PERIOD_ERROR_MODE_FLEXIBLE	Bit period error Flexible Mode

Bit Timing Error Mode

CEC_BIT_TIMING_ERROR_MODE_STANDARD	Bit timing error Standard Mode
CEC_BIT_TIMING_ERROR_MODE_ERRORFREE	Bit timing error Free Mode

CEC Exported Macros

__HAL_CEC_RESET_HANDLE_STATE

Description:

- Reset CEC handle state.

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.
- __INTERRUPT__: specifies the interrupt to check.
  - CEC_FLAG_TERR: Tx Error
  - CEC_FLAG_TBTF: Tx Block Transfer Finished
  - CEC_FLAG_RERR: Rx Error
  - CEC_FLAG_RBTF: Rx Block Transfer Finished

Return value:

- ITStatus:

__HAL_CEC_CLEAR_FLAG

Description:

- Clears the CEC's pending flags.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - CEC_CSR_TERR: Tx Error
  - CEC_CSR_TBTF: Tx Block Transfer Finished
  - CEC_CSR_RERR: Rx Error
  - CEC_CSR_RBTF: Rx Block Transfer Finished

Return value:

- none:

__HAL_CEC_ENABLE_IT

Description:

- Enables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: The CEC interrupt to enable. This parameter can be:
  - CEC_IT_IE : Interrupt Enable

Return value:

- none:

__HAL_CEC_DISABLE_IT

Description:

- Disables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: The CEC interrupt to enable. This parameter can be:
  - CEC_IT_IE : Interrupt

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__: The CEC interrupt to enable. This parameter can be:
  - CEC_IT_IE : Interrupt 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:

__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.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __ADDRESS__: Own Address value.

Return value:

- none:

Flags definition

CEC_FLAG_TSOM

CEC_FLAG_TEOM

CEC_FLAG_TERR

CEC_FLAG_TBTRF

CEC_FLAG_RSOM

CEC_FLAG_REOM

CEC_FLAG_RERR

CEC_FLAG_RBTF

Initiator logical address position in message header

CEC_INITIATOR_LSB_POS

Interrupts definition

CEC_IT_IE

CEC Private Constants

CEC_CFGR_FIELDS

CEC_FLAG_TRANSMIT_MASK

CEC_FLAG_RECEIVE_MASK

CEC_ESR_ALL_ERROR

CEC_RXXFERSIZE_INITIALIZE — Value used to initialise the RxXferSize of the handle

IS_CEC_BIT_TIMING_ERROR_MODE

IS_CEC_BIT_PERIOD_ERROR_MODE

IS_CEC_OAR_ADDRESS

Description:

- Check CEC device Own Address Register (OAR) setting.

Parameters:

- __ADDRESS__: CEC own address.

Return value:

- Test: result (TRUE or FALSE).

IS_CEC_ADDRESS

Description:

- Check CEC initiator or destination logical address setting.

Parameters:

- __ADDRESS__: CEC initiator or logical address.

Return value:

- Test: result (TRUE or FALSE).

IS_CEC_MSGSIZE

Description:

- Check CEC message size.

Parameters:

- __SIZE__: CEC message size.

Return value:

- Test: result (TRUE or FALSE).

9 HAL CORTEX Generic Driver

9.1 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

9.1.1 Initialization and de-initialization functions

This section provide the Cortex HAL driver functions allowing to configure Interrupts Systick functionalities

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

9.1.2 Peripheral Control functions

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

- *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()*

9.1.3 HAL_NVIC_SetPriorityGrouping

Function Name	**void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)**
Function Description	Sets the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.
Parameters	• **PriorityGroup:** The priority grouping bits length. This parameter can be one of the following values: NVIC_PRIORITYGROUP_0: 0 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

Return values	• None
Notes	• When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the subpriority.

9.1.4 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 (stm32f10xxx.h)) • **PreemptPriority:** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority • **SubPriority:** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.
Return values	• None

9.1.5 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 (stm32f10xxx.h))
Return values	• None
Notes	• To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

9.1.6 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 (stm32f10xxx.h))
Return values	• None

9.1.7 HAL_NVIC_SystemReset

Function Name	**void HAL_NVIC_SystemReset (void)**
Function Description	Initiates a system reset request to reset the MCU.
Return values	• None

9.1.8 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.

9.1.9 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)

9.1.10 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 (stm32f10xxx.h)) • **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values: 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 • **pPreemptPriority:** Pointer on the Preemptive priority value (starting from 0). • **pSubPriority:** Pointer on the Subpriority value (starting from

0).

Return values • None

9.1.11 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 (stm32f10xxx.h))
Return values	• None

9.1.12 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 (stm32f10xxx.h))
Return values	• status - 0 Interrupt status is not pending. 1 Interrupt status is pending.

9.1.13 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 (stm32f10xxx.h))
Return values	• None

9.1.14 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 (stm32f10xxx.h))
Return values	• status - 0 Interrupt status is not pending. 1 Interrupt status is pending.

9.1.15 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

9.1.16 HAL_SYSTICK_IRQHandler

Function Name	**void HAL_SYSTICK_IRQHandler (void)**
Function Description	This function handles SYSTICK interrupt request.
Return values	• None

9.1.17 HAL_SYSTICK_Callback

Function Name	**void HAL_SYSTICK_Callback (void)**
Function Description	SYSTICK callback.
Return values	• None

9.2 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

9.2.1 CORTEX

CORTEX

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 Preemption Priority Group

IS_NVIC_PRIORITY_GROUP

IS_NVIC_PREEMPTION_PRIORITY

IS_NVIC_SUB_PRIORITY

IS_NVIC_DEVICE_IRQ

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

CORTEX SysTick clock source

__HAL_CORTEX_SYSTICKCLK_CONFIG

Description:

- Configures the SysTick clock source.

Parameters:

- __CLKSRC__: 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 value:

- None:

CORTEX SysTick clock source

IS_SYSTICK_CLK_SOURCE

10 HAL CRC Generic Driver

10.1 CRC Firmware driver registers structures

10.1.1 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32f1xx_hal_crc.h

Data Fields

- ***CRC_TypeDef * Instance***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance*** Register base address
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock*** CRC locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State*** CRC communication state

10.2 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

10.2.1 How to use this driver

The CRC HAL driver can be used as follows:

1. Enable CRC AHB clock using __HAL_RCC_CRC_CLK_ENABLE();
2. Use HAL_CRC_Accumulate() function to compute the CRC value of a 32-bit data buffer using combination of the previous CRC value and the new one.
3. Use HAL_CRC_Calculate() function to compute the CRC Value of a new 32-bit data buffer. This function resets the CRC computation unit before starting the computation to avoid getting wrong CRC values.

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

10.2.3 Peripheral Control functions

This section provides functions allowing to:

- Compute the 32-bit CRC value of 32-bit data buffer, using combination of the previous CRC value and the new one.
- Compute the 32-bit CRC value of 32-bit data buffer, independently of the previous CRC value.
- ***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.

- ***HAL_CRC_GetState()***

10.2.5 HAL_CRC_Init

Function Name	**HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)**
Function Description	Initializes the CRC according to the specified parameters in the CRC_InitTypeDef and creates the associated handle.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• HAL status

10.2.6 HAL_CRC_DeInit

Function Name	**HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)**
Function Description	DeInitializes the CRC peripheral.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• HAL status

10.2.7 HAL_CRC_MspInit

Function Name	**void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)**
Function Description	Initializes the CRC MSP.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• None

10.2.8 HAL_CRC_MspDeInit

Function Name	**void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)**
Function Description	DeInitializes the CRC MSP.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	• None

10.2.9 HAL_CRC_Accumulate

Function Name	**uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)**
Function Description	Computes the 32-bit CRC of 32-bit data buffer using combination of the previous CRC value and the new one.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC • **pBuffer:** pointer to the buffer containing the data to be computed • **BufferLength:** length of the buffer to be computed (defined in word, 4 bytes)
Return values	• 32-bit CRC

10.2.10 HAL_CRC_Calculate

Function Name	**uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)**
Function Description	Computes the 32-bit CRC of 32-bit data buffer independently of the previous CRC value.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC • **pBuffer:** Pointer to the buffer containing the data to be computed • **BufferLength:** Length of the buffer to be computed (defined in word, 4 bytes)
Return values	• 32-bit CRC

10.2.11 HAL_CRC_GetState

Function Name	**HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)**
Function Description	Returns the CRC state.
Parameters	• **hcrc:** pointer to a CRC_HandleTypeDef structure that

contains the configuration information for CRC

Return values
- HAL state

10.3 CRC Firmware driver defines

The following section lists the various define and macros of the module.

10.3.1 CRC

CRC

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:

- Resets CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None:

__HAL_CRC_SET_IDR

Description:

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

Parameters:

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

Return value:

- None:

__HAL_CRC_GET_IDR

Description:

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

Parameters:

- __HANDLE__: CRC handle

Return value:

- 8-bit: value of the ID register

11 HAL DAC Generic Driver

11.1 DAC Firmware driver registers structures

11.1.1 DAC_HandleTypeDef

DAC_HandleTypeDef is defined in the stm32f1xx_hal_dac.h

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

11.1.2 DAC_ChannelConfTypeDef

DAC_ChannelConfTypeDef is defined in the stm32f1xx_hal_dac.h

Data Fields

- ***uint32_t DAC_Trigger***
- ***uint32_t DAC_OutputBuffer***

Field Documentation

- ***uint32_t DAC_ChannelConfTypeDef::DAC_Trigger*** Specifies the external trigger for the selected DAC channel. This parameter can be a value of ***DACEx_trigger_selection*** Note: For STM32F100x high-density value line devices, additional trigger sources are available.
- ***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***

11.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

11.2.1 DAC Peripheral features

DAC Channels

The device integrates two 12-bit Digital Analog Converters that can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 (PA4) as output
2. DAC channel2 with DAC_OUT2 (PA5) as output

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM4, TIM6, TIM7 For STM32F10x connectivity line devices and STM32F100x devices: TIM3 For STM32F10x high-density and XL-density devices: TIM8 For STM32F100x high-density value line devices: TIM15 as replacement of TIM5. (DAC_TRIGGER_T2_TRGO, DAC_TRIGGER_T4_TRGO...)
3. Software using DAC_TRIGGER_SOFTWARE

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;

Refer to the device datasheet for more details about output impedance value with and without output buffer.

DAC connect feature

Each DAC channel can be connected internally. To connect, use sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_ENABLE;

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel1 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 wave generation feature

Both DAC channels can be used to generate

1. Noise wave using HAL_DACEx_NoiseWaveGenerate()
2. Triangle wave using HAL_DACEx_TriangleWaveGenerate()

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 correspondance

The analog output voltage on each DAC channel pin is determined by the following equation:

DAC_OUTx = VREF+ * DOR / 4095

- with DOR is the Data Output Register

VEF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

A DMA1 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 requests are enabled using HAL_DAC_Start_DMA()

DMA requests are mapped as following:

1. DAC channel1 : For STM32F100x low-density, medium-density, high-density with DAC DMA remap: mapped on DMA1 channel3 which must be already configured For STM32F100x high-density without DAC DMA remap and other STM32F1 devices: mapped on DMA2 channel3 which must be already configured
2. DAC channel2 : For STM32F100x low-density, medium-density, high-density with DAC DMA remap: mapped on DMA1 channel4 which must be already configured For STM32F100x high-density without DAC DMA remap and other STM32F1 devices: mapped on DMA2 channel4 which must be already configured

11.2.2 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

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_DACEx_ConvHalfCpltCallbackCh1()or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1 or HAL_DAC_ConvHalfCpltCallbackCh2
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1()or HAL_DAC_ConvCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1 or HAL_DAC_ConvCpltCallbackCh2
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() or HAL_DACEx_ErrorCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1 or HAL_DACEx_ErrorCallbackCh2
- For STM32F100x devices with specific feature: DMA underrun. In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1()or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1 or HAL_DACEx_DMAUnderrunCallbackCh2 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 (For STM32F100x devices with specific feature: DMA underrun)
- __HAL_DAC_DISABLE : Disable the DAC peripheral (For STM32F100x devices with specific feature: DMA underrun)
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags (For STM32F100x devices with specific feature: DMA underrun)
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status (For STM32F100x devices with specific feature: DMA underrun)

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

11.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.
- ***HAL_DAC_Init()***

- *HAL_DAC_DeInit()*
- *HAL_DAC_MspInit()*
- *HAL_DAC_MspDeInit()*

11.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- *HAL_DAC_Start()*
- *HAL_DAC_Stop()*
- *HAL_DAC_Start_DMA()*
- *HAL_DAC_Stop_DMA()*
- *HAL_DAC_GetValue()*
- *HAL_DAC_ConvCpltCallbackCh1()*
- *HAL_DAC_ConvHalfCpltCallbackCh1()*
- *HAL_DAC_ErrorCallbackCh1()*
- *HAL_DAC_SetValue()*

11.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.
- *HAL_DAC_ConfigChannel()*
- *HAL_DAC_SetValue()*

11.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.
- *HAL_DAC_GetState()*
- *HAL_DAC_GetError()*
- *HAL_DAC_ConvCpltCallbackCh1()*
- *HAL_DAC_ConvHalfCpltCallbackCh1()*
- *HAL_DAC_ErrorCallbackCh1()*

11.2.7 HAL_DAC_Init

Function Name	**HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)**
Function Description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.

Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• HAL status

11.2.8 HAL_DAC_DeInit

Function Name	**HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)**
Function Description	Deinitializes 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

11.2.9 HAL_DAC_MspInit

Function Name	**void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)**
Function Description	Initializes the DAC MSP.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None

11.2.10 HAL_DAC_MspDeInit

Function Name	**void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)**
Function Description	DeInitializes the DAC MSP.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None

11.2.11 HAL_DAC_Start

Function Name	**HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function Description	Enables DAC and starts conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selected DAC_CHANNEL_2: DAC Channel2 selected

Return values	• HAL status

11.2.12 HAL_DAC_Stop

Function Name	**HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function Description	Disables DAC and stop conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selected DAC_CHANNEL_2: DAC Channel2 selected
Return values	• HAL status

11.2.13 HAL_DAC_Start_DMA

Function Name	**HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)**
Function Description	Enables DAC and starts conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selected DAC_CHANNEL_2: DAC Channel2 selected • **pData:** The destination peripheral Buffer address. • **Length:** The length of data to be transferred from memory to DAC peripheral • **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected
Return values	• HAL status

11.2.14 HAL_DAC_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function Description	Disables DAC and stop conversion of channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC

	Channel1 selected DAC_CHANNEL_2: DAC Channel2 selected
Return values	• HAL status

11.2.15 HAL_DAC_GetValue

Function Name	**uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)**
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selected DAC_CHANNEL_2: DAC Channel2 selected
Return values	• The selected DAC channel data output value.

11.2.16 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

11.2.17 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

11.2.18 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

11.2.19 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

11.2.20 HAL_DAC_ConfigChannel

Function Name	**HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)**
Function Description	Configures the selected DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **sConfig:** DAC configuration structure. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selected DAC_CHANNEL_2: DAC Channel2 selected
Return values	• HAL status

11.2.21 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

11.2.22 HAL_DAC_GetState

Function Name	**HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)**
Function Description	return the DAC state
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• HAL state

11.2.23 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

11.2.24 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

11.2.25 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

11.2.26 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

11.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

11.3.1 DAC

DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC data alignement

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE	No error
HAL_DAC_ERROR_DMAUNDERRUNCH1	DAC channel1 DMA underrun error
HAL_DAC_ERROR_DMAUNDERRUNCH2	DAC channel2 DMA underrun error
HAL_DAC_ERROR_DMA	DMA error

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 output buffer

DAC_OUTPUTBUFFER_ENABLE

DAC_OUTPUTBUFFER_DISABLE

DAC Private Macros

IS_DAC_OUTPUT_BUFFER_STATE

IS_DAC_CHANNEL

IS_DAC_ALIGN

IS_DAC_DATA

DAC_DHR12R1_ALIGNMENT

DAC_DHR12R2_ALIGNMENT

DAC_DHR12RD_ALIGNMENT

12 HAL DAC Extension Driver

12.1 DACEx Firmware driver API description

The following section lists the various functions of the DACEx library.

12.1.1 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.

12.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.
- ***HAL_DACEx_DualGetValue()***
- ***HAL_DACEx_TriangleWaveGenerate()***
- ***HAL_DACEx_NoiseWaveGenerate()***
- ***HAL_DACEx_DualSetValue()***
- ***HAL_DACEx_ConvCpltCallbackCh2()***
- ***HAL_DACEx_ConvHalfCpltCallbackCh2()***
- ***HAL_DACEx_ErrorCallbackCh2()***

12.1.3 HAL_DACEx_DualGetValue

Function Name	**uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)**
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• The selected DAC channel data output value.

12.1.4 HAL_DACEx_TriangleWaveGenerate

Function Name	**HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)**
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2 • **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values: DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1 DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3 DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7 DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15 DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31 DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63 DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127 DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255 DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511 DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023 DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047 DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095
Return values	• HAL status

12.1.5 HAL_DACEx_NoiseWaveGenerate

Function Name	**HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)**
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2 • **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values: DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR

bit[3:0] for noise wave generation
DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation
DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation
DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values
- HAL status

12.1.6 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.

12.1.7 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

12.1.8 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

12.1.9 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

12.2 DACEx Firmware driver defines

The following section lists the various define and macros of the module.

12.2.1 DACEx

DACEx

DACEx lfsrunmask triangleamplitude

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

DACEx Private Macros

IS_DAC_TRIGGER

IS_DAC_GENERATE_WAVE

IS_DAC_LFSR_UNMASK_TRIANGLE_AMPLITUDE

IS_DAC_WAVE

DAC trigger selection

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
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_T5_TRGO	TIM5 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_EXT_IT9	EXTI Line9 event selected as external conversion trigger

	for DAC channel
DAC_TRIGGER_SOFTWARE	Conversion started by software trigger for DAC channel
DAC_TRIGGER_T8_TRGO	TIM8 TRGO selected as external conversion trigger for DAC channel

DACEx wave generation

DAC_WAVEGENERATION_NOISE

DAC_WAVEGENERATION_TRIANGLE

DAC_WAVE_NOISE

DAC_WAVE_TRIANGLE

13 HAL DMA Generic Driver

13.1 DMA Firmware driver registers structures

13.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32f1xx_hal_dma.h

Data Fields

- ***uint32_t Direction***
- ***uint32_t PeriphInc***
- ***uint32_t MemInc***
- ***uint32_t PeriphDataAlignment***
- ***uint32_t MemDataAlignment***
- ***uint32_t Mode***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Direction*** Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_Data_transfer_direction***
- ***uint32_t DMA_InitTypeDef::PeriphInc*** Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of ***DMA_Peripheral_incremented_mode***
- ***uint32_t DMA_InitTypeDef::MemInc*** Specifies whether the memory address register should be incremented or not. This parameter can be a value of ***DMA_Memory_incremented_mode***
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment*** Specifies the Peripheral data width. This parameter can be a value of ***DMA_Peripheral_data_size***
- ***uint32_t DMA_InitTypeDef::MemDataAlignment*** Specifies the Memory data width. This parameter can be a value of ***DMA_Memory_data_size***
- ***uint32_t DMA_InitTypeDef::Mode*** Specifies the operation mode of the DMAy Channelx. This parameter can be a value of ***DMA_mode***
  Note: The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel
- ***uint32_t DMA_InitTypeDef::Priority*** Specifies the software priority for the DMAy Channelx. This parameter can be a value of ***DMA_Priority_level***

13.1.2 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32f1xx_hal_dma.h

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***HAL_DMA_StateTypeDef State***
- ***void * Parent***
- ***void(* XferCpltCallback***

- ***void(* XferHalfCpltCallback***
- ***void(* XferErrorCallback***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance*** Register base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init*** DMA communication parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock*** DMA locking object
- ***HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State*** DMA transfer state
- ***void* __DMA_HandleTypeDef::Parent*** Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)*** DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)*** DMA Half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)*** DMA transfer error callback
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode*** DMA Error code

13.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

13.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Channel (except for internal SRAM / FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests .
2. For a given Channel, program the required configuration through the following parameters: Transfer Direction, Source and Destination data formats, Circular or Normal mode, Channel Priority level, Source and Destination Increment mode, using HAL_DMA_Init() function.
3. Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
4. Use HAL_DMA_Abort() function to abort the current transfer In Memory-to-Memory transfer mode, Circular mode is not allowed.

Polling mode IO operation

- Use HAL_DMA_Start() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred
- Use HAL_DMA_PollForTransfer() to poll for the end of current transfer, in this case a fixed Timeout can be configured by User depending from his application.

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()

- Use HAL_DMA_Start_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL_DMAy_Channelx_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).

DMA HAL driver macros list

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

- __HAL_DMA_ENABLE: Enable the specified DMA Channel.
- __HAL_DMA_DISABLE: Disable the specified DMA Channel.
- __HAL_DMA_GET_FLAG: Get the DMA Channel pending flags.
- __HAL_DMA_CLEAR_FLAG: Clear the DMA Channel pending flags.
- __HAL_DMA_ENABLE_IT: Enable the specified DMA Channel interrupts.
- __HAL_DMA_DISABLE_IT: Disable the specified DMA Channel interrupts.
- __HAL_DMA_GET_IT_SOURCE: Check whether the specified DMA Channel interrupt has occurred or not.

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

13.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Channel source and destination addresses, incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Channel priority value.

The HAL_DMA_Init() function follows the DMA configuration procedures as described in reference manual.

- ***HAL_DMA_Init()***
- ***HAL_DMA_DeInit()***

13.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request
- ***HAL_DMA_Start()***
- ***HAL_DMA_Start_IT()***
- ***HAL_DMA_Abort()***
- ***HAL_DMA_PollForTransfer()***
- ***HAL_DMA_IRQHandler()***

13.2.4 State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code
- *HAL_DMA_GetState()*
- *HAL_DMA_GetError()*

13.2.5 HAL_DMA_Init

Function Name	**HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)**
Function Description	Initializes 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 Channel.
Return values	• HAL status

13.2.6 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 Channel.
Return values	• HAL status

13.2.7 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 Channel. • **SrcAddress:** The source memory Buffer address • **DstAddress:** The destination memory Buffer address • **DataLength:** The length of data to be transferred from source to destination
Return values	• HAL status

13.2.8 HAL_DMA_Start_IT

Function Name	**HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)**
Function Description	Start the DMA Transfer with interrupt enabled.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **SrcAddress:** The source memory Buffer address • **DstAddress:** The destination memory Buffer address • **DataLength:** The length of data to be transferred from source to destination
Return values	• HAL status

13.2.9 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 Channel.
Return values	• HAL status
Notes	• After disabling a DMA Channel, a check for wait until the DMA Channel is effectively disabled is added. If a Channel is disabled while a data transfer is ongoing, the current data will be transferred and the Channel will be effectively disabled only after the transfer of this single data is finished.

13.2.10 HAL_DMA_PollForTransfer

Function Name	**HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)**
Function Description	Polling for transfer complete.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • **CompleteLevel:** Specifies the DMA level complete. • **Timeout:** Timeout duration.
Return values	• HAL status

13.2.11 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 Channel.
Return values	• None

13.2.12 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 Channel.
Return values	• HAL state

13.2.13 HAL_DMA_GetError

Function Name	**uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)**
Function Description	Return the DMA error code.
Parameters	• **hdma:** : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	• DMA Error Code

13.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

13.3.1 DMA

DMA

DMA Data buffer size

IS_DMA_BUFFER_SIZE

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

IS_DMA_DIRECTION

DMA Error Codes

HAL_DMA_ERROR_NONE	No error
HAL_DMA_ERROR_TE	Transfer error
HAL_DMA_ERROR_TIMEOUT	Timeout error

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: DMA handle.

Return value:

- None:

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None.:

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: DMA handle

Return value:

- None.:

__HAL_DMA_ENABLE_IT

Description:

- Enables the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  – DMA_IT_TC: Transfer complete interrupt mask
  – DMA_IT_HT: Half transfer complete interrupt mask
  – DMA_IT_TE: Transfer error interrupt mask

Return value:

- None:

__HAL_DMA_DISABLE_IT

Description:

- Disables the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

Return value:

- None:

__HAL_DMA_GET_IT_SOURCE

Description:

- Checks whether the specified DMA Channel interrupt has occurred or not.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt source to check. This parameter can be one of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

Return value:

- The: state of DMA_IT (SET or RESET).

DMA flag definitions

DMA_FLAG_GL1

DMA_FLAG_TC1

DMA_FLAG_HT1

DMA_FLAG_TE1

DMA_FLAG_GL2

DMA_FLAG_TC2

DMA_FLAG_HT2

DMA_FLAG_TE2

DMA_FLAG_GL3

DMA_FLAG_TC3

DMA_FLAG_HT3

DMA_FLAG_TE3

DMA_FLAG_GL4

DMA_FLAG_TC4

DMA_FLAG_HT4

DMA_FLAG_TE4

DMA_FLAG_GL5

DMA_FLAG_TC5

DMA_FLAG_HT5

DMA_FLAG_TE5

DMA_FLAG_GL6

DMA_FLAG_TC6

DMA_FLAG_HT6

DMA_FLAG_TE6

DMA_FLAG_GL7

DMA_FLAG_TC7

DMA_FLAG_HT7

DMA_FLAG_TE7

DMA interrupt enable definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA Memory data size

DMA_MDATAALIGN_BYTE	Memory data alignment : Byte
DMA_MDATAALIGN_HALFWORD	Memory data alignment : HalfWord
DMA_MDATAALIGN_WORD	Memory data alignment : Word
IS_DMA_MEMORY_DATA_SIZE	

DMA Memory incremented mode

DMA_MINC_ENABLE	Memory increment mode Enable
DMA_MINC_DISABLE	Memory increment mode Disable
IS_DMA_MEMORY_INC_STATE	

DMA mode

DMA_NORMAL	Normal Mode
DMA_CIRCULAR	Circular Mode

IS_DMA_MODE

DMA Peripheral data size

DMA_PDATAALIGN_BYTE	Peripheral data alignment : Byte
DMA_PDATAALIGN_HALFWORD	Peripheral data alignment : HalfWord
DMA_PDATAALIGN_WORD	Peripheral data alignment : Word
IS_DMA_PERIPHERAL_DATA_SIZE	

DMA Peripheral incremented mode

DMA_PINC_ENABLE	Peripheral increment mode Enable
DMA_PINC_DISABLE	Peripheral increment mode Disable
IS_DMA_PERIPHERAL_INC_STATE	

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
IS_DMA_PRIORITY	

DMA Private Constants

HAL_TIMEOUT_DMA_ABORT

14 HAL DMA Extension Driver

14.1 DMAEx Firmware driver defines

The following section lists the various define and macros of the module.

14.1.1 DMAEx

DMAEx

DMAEx High density and XL density product devices

__HAL_DMA_GET_TC_FLAG_INDEX

__HAL_DMA_GET_HT_FLAG_INDEX

Description:

- Returns the current DMA Channel half transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified half transfer complete flag index.

__HAL_DMA_GET_TE_FLAG_INDEX

Description:

- Returns the current DMA Channel transfer error flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_FLAG

Description:

- Get the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  – DMA_FLAG_TCx: Transfer complete flag
  – DMA_FLAG_HTx: Half transfer complete flag
  – DMA_FLAG_TEx: Transfer error flag Where x can be 1_7 or 1_5 (depending on DMA1 or DMA2) to select the DMA Channel flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DMA_CLEAR_FLAG

Description:

- Clears the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA_FLAG_TCx: Transfer complete flag
  - DMA_FLAG_HTx: Half transfer complete flag
  - DMA_FLAG_TEx: Transfer error flag Where x can be 1_7 or 1_5 (depending on DMA1 or DMA2) to select the DMA Channel flag.

Return value:

- None:

15 HAL ETH Generic Driver

15.1 ETH Firmware driver registers structures

15.1.1 ETH_InitTypeDef

ETH_InitTypeDef is defined in the stm32f1xx_hal_eth.h

Data Fields

- ***uint32_t AutoNegotiation***
- ***uint32_t Speed***
- ***uint32_t DuplexMode***
- ***uint16_t PhyAddress***
- ***uint8_t * MACAddr***
- ***uint32_t RxMode***
- ***uint32_t ChecksumMode***
- ***uint32_t MediaInterface***

Field Documentation

- ***uint32_t ETH_InitTypeDef::AutoNegotiation*** Selects or not the AutoNegotiation mode for the external PHY The AutoNegotiation allows an automatic setting of the Speed (10/100Mbps) and the mode (half/full-duplex). This parameter can be a value of ***ETH_AutoNegotiation***
- ***uint32_t ETH_InitTypeDef::Speed*** Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of ***ETH_Speed***
- ***uint32_t ETH_InitTypeDef::DuplexMode*** Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode This parameter can be a value of ***ETH_Duplex_Mode***
- ***uint16_t ETH_InitTypeDef::PhyAddress*** Ethernet PHY address. This parameter must be a number between Min_Data = 0 and Max_Data = 32
- ***uint8_t* ETH_InitTypeDef::MACAddr*** MAC Address of used Hardware: must be pointer on an array of 6 bytes
- ***uint32_t ETH_InitTypeDef::RxMode*** Selects the Ethernet Rx mode: Polling mode, Interrupt mode. This parameter can be a value of ***ETH_Rx_Mode***
- ***uint32_t ETH_InitTypeDef::ChecksumMode*** Selects if the checksum is check by hardware or by software. This parameter can be a value of ***ETH_Checksum_Mode***
- ***uint32_t ETH_InitTypeDef::MediaInterface*** Selects the media-independent interface or the reduced media-independent interface. This parameter can be a value of ***ETH_Media_Interface***

15.1.2 ETH_MACInitTypeDef

ETH_MACInitTypeDef is defined in the stm32f1xx_hal_eth.h

Data Fields

- ***uint32_t Watchdog***
- ***uint32_t Jabber***
- ***uint32_t InterFrameGap***
- ***uint32_t CarrierSense***
- ***uint32_t ReceiveOwn***

- ***uint32_t LoopbackMode***
- ***uint32_t ChecksumOffload***
- ***uint32_t RetryTransmission***
- ***uint32_t AutomaticPadCRCStrip***
- ***uint32_t BackOffLimit***
- ***uint32_t DeferralCheck***
- ***uint32_t ReceiveAll***
- ***uint32_t SourceAddrFilter***
- ***uint32_t PassControlFrames***
- ***uint32_t BroadcastFramesReception***
- ***uint32_t DestinationAddrFilter***
- ***uint32_t PromiscuousMode***
- ***uint32_t MulticastFramesFilter***
- ***uint32_t UnicastFramesFilter***
- ***uint32_t HashTableHigh***
- ***uint32_t HashTableLow***
- ***uint32_t PauseTime***
- ***uint32_t ZeroQuantaPause***
- ***uint32_t PauseLowThreshold***
- ***uint32_t UnicastPauseFrameDetect***
- ***uint32_t ReceiveFlowControl***
- ***uint32_t TransmitFlowControl***
- ***uint32_t VLANTagComparison***
- ***uint32_t VLANTagIdentifier***

Field Documentation

- ***uint32_t ETH_MACInitTypeDef::Watchdog*** Selects or not the Watchdog timer When enabled, the MAC allows no more then 2048 bytes to be received. When disabled, the MAC can receive up to 16384 bytes. This parameter can be a value of ***ETH_Watchdog***
- ***uint32_t ETH_MACInitTypeDef::Jabber*** Selects or not Jabber timer When enabled, the MAC allows no more then 2048 bytes to be sent. When disabled, the MAC can send up to 16384 bytes. This parameter can be a value of ***ETH_Jabber***
- ***uint32_t ETH_MACInitTypeDef::InterFrameGap*** Selects the minimum IFG between frames during transmission. This parameter can be a value of ***ETH_Inter_Frame_Gap***
- ***uint32_t ETH_MACInitTypeDef::CarrierSense*** Selects or not the Carrier Sense. This parameter can be a value of ***ETH_Carrier_Sense***
- ***uint32_t ETH_MACInitTypeDef::ReceiveOwn*** Selects or not the ReceiveOwn, ReceiveOwn allows the reception of frames when the TX_EN signal is asserted in Half-Duplex mode. This parameter can be a value of ***ETH_Receive_Own***
- ***uint32_t ETH_MACInitTypeDef::LoopbackMode*** Selects or not the internal MAC MII Loopback mode. This parameter can be a value of ***ETH_Loop_Back_Mode***
- ***uint32_t ETH_MACInitTypeDef::ChecksumOffload*** Selects or not the IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. This parameter can be a value of ***ETH_Checksum_Offload***
- ***uint32_t ETH_MACInitTypeDef::RetryTransmission*** Selects or not the MAC attempt retries transmission, based on the settings of BL, when a collision occurs (Half-Duplex mode). This parameter can be a value of ***ETH_Retry_Transmission***
- ***uint32_t ETH_MACInitTypeDef::AutomaticPadCRCStrip*** Selects or not the Automatic MAC Pad/CRC Stripping. This parameter can be a value of ***ETH_Automatic_Pad_CRC_Strip***

- ***uint32_t ETH_MACInitTypeDef::BackOffLimit*** Selects the BackOff limit value. This parameter can be a value of ***ETH_Back_Off_Limit***
- ***uint32_t ETH_MACInitTypeDef::DeferralCheck*** Selects or not the deferral check function (Half-Duplex mode). This parameter can be a value of ***ETH_Deferral_Check***
- ***uint32_t ETH_MACInitTypeDef::ReceiveAll*** Selects or not all frames reception by the MAC (No filtering). This parameter can be a value of ***ETH_Receive_All***
- ***uint32_t ETH_MACInitTypeDef::SourceAddrFilter*** Selects the Source Address Filter mode. This parameter can be a value of ***ETH_Source_Addr_Filter***
- ***uint32_t ETH_MACInitTypeDef::PassControlFrames*** Sets the forwarding mode of the control frames (including unicast and multicast PAUSE frames) This parameter can be a value of ***ETH_Pass_Control_Frames***
- ***uint32_t ETH_MACInitTypeDef::BroadcastFramesReception*** Selects or not the reception of Broadcast Frames. This parameter can be a value of ***ETH_Broadcast_Frames_Reception***
- ***uint32_t ETH_MACInitTypeDef::DestinationAddrFilter*** Sets the destination filter mode for both unicast and multicast frames. This parameter can be a value of ***ETH_Destination_Addr_Filter***
- ***uint32_t ETH_MACInitTypeDef::PromiscuousMode*** Selects or not the Promiscuous Mode This parameter can be a value of ***ETH_Promiscuous_Mode***
- ***uint32_t ETH_MACInitTypeDef::MulticastFramesFilter*** Selects the Multicast Frames filter mode: None/HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of ***ETH_Multicast_Frames_Filter***
- ***uint32_t ETH_MACInitTypeDef::UnicastFramesFilter*** Selects the Unicast Frames filter mode: HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of ***ETH_Unicast_Frames_Filter***
- ***uint32_t ETH_MACInitTypeDef::HashTableHigh*** This field holds the higher 32 bits of Hash table. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFFFFFF
- ***uint32_t ETH_MACInitTypeDef::HashTableLow*** This field holds the lower 32 bits of Hash table. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFFFFFF
- ***uint32_t ETH_MACInitTypeDef::PauseTime*** This field holds the value to be used in the Pause Time field in the transmit control frame. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFF
- ***uint32_t ETH_MACInitTypeDef::ZeroQuantaPause*** Selects or not the automatic generation of Zero-Quanta Pause Control frames. This parameter can be a value of ***ETH_Zero_Quanta_Pause***
- ***uint32_t ETH_MACInitTypeDef::PauseLowThreshold*** This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Frame. This parameter can be a value of ***ETH_Pause_Low_Threshold***
- ***uint32_t ETH_MACInitTypeDef::UnicastPauseFrameDetect*** Selects or not the MAC detection of the Pause frames (with MAC Address0 unicast address and unique multicast address). This parameter can be a value of ***ETH_Unicast_Pause_Frame_Detect***
- ***uint32_t ETH_MACInitTypeDef::ReceiveFlowControl*** Enables or disables the MAC to decode the received Pause frame and disable its transmitter for a specified time (Pause Time) This parameter can be a value of ***ETH_Receive_Flow_Control***
- ***uint32_t ETH_MACInitTypeDef::TransmitFlowControl*** Enables or disables the MAC to transmit Pause frames (Full-Duplex mode) or the MAC back-pressure operation (Half-Duplex mode) This parameter can be a value of ***ETH_Transmit_Flow_Control***
- ***uint32_t ETH_MACInitTypeDef::VLANTagComparison*** Selects the 12-bit VLAN identifier or the complete 16-bit VLAN tag for comparison and filtering. This parameter can be a value of ***ETH_VLAN_Tag_Comparison***

- ***uint32_t ETH_MACInitTypeDef::VLANTagIdentifier*** Holds the VLAN tag identifier for receive frames

15.1.3 ETH_DMAInitTypeDef

ETH_DMAInitTypeDef is defined in the stm32f1xx_hal_eth.h

Data Fields

- ***uint32_t DropTCPIPChecksumErrorFrame***
- ***uint32_t ReceiveStoreForward***
- ***uint32_t FlushReceivedFrame***
- ***uint32_t TransmitStoreForward***
- ***uint32_t TransmitThresholdControl***
- ***uint32_t ForwardErrorFrames***
- ***uint32_t ForwardUndersizedGoodFrames***
- ***uint32_t ReceiveThresholdControl***
- ***uint32_t SecondFrameOperate***
- ***uint32_t AddressAlignedBeats***
- ***uint32_t FixedBurst***
- ***uint32_t RxDMABurstLength***
- ***uint32_t TxDMABurstLength***
- ***uint32_t DescriptorSkipLength***
- ***uint32_t DMAArbitration***

Field Documentation

- ***uint32_t ETH_DMAInitTypeDef::DropTCPIPChecksumErrorFrame*** Selects or not the Dropping of TCP/IP Checksum Error Frames. This parameter can be a value of ***ETH_Drop_TCP_IP_Checksum_Error_Frame***
- ***uint32_t ETH_DMAInitTypeDef::ReceiveStoreForward*** Enables or disables the Receive store and forward mode. This parameter can be a value of ***ETH_Receive_Store_Forward***
- ***uint32_t ETH_DMAInitTypeDef::FlushReceivedFrame*** Enables or disables the flushing of received frames. This parameter can be a value of ***ETH_Flush_Received_Frame***
- ***uint32_t ETH_DMAInitTypeDef::TransmitStoreForward*** Enables or disables Transmit store and forward mode. This parameter can be a value of ***ETH_Transmit_Store_Forward***
- ***uint32_t ETH_DMAInitTypeDef::TransmitThresholdControl*** Selects or not the Transmit Threshold Control. This parameter can be a value of ***ETH_Transmit_Threshold_Control***
- ***uint32_t ETH_DMAInitTypeDef::ForwardErrorFrames*** Selects or not the forward to the DMA of erroneous frames. This parameter can be a value of ***ETH_Forward_Error_Frames***
- ***uint32_t ETH_DMAInitTypeDef::ForwardUndersizedGoodFrames*** Enables or disables the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC) This parameter can be a value of ***ETH_Forward_Undersized_Good_Frames***
- ***uint32_t ETH_DMAInitTypeDef::ReceiveThresholdControl*** Selects the threshold level of the Receive FIFO. This parameter can be a value of ***ETH_Receive_Threshold_Control***

- ***uint32_t ETH_DMAInitTypeDef::SecondFrameOperate*** Selects or not the Operate on second frame mode, which allows the DMA to process a second frame of Transmit data even before obtaining the status for the first frame. This parameter can be a value of ***ETH_Second_Frame_Operate***
- ***uint32_t ETH_DMAInitTypeDef::AddressAlignedBeats*** Enables or disables the Address Aligned Beats. This parameter can be a value of ***ETH_Address_Aligned_Beats***
- ***uint32_t ETH_DMAInitTypeDef::FixedBurst*** Enables or disables the AHB Master interface fixed burst transfers. This parameter can be a value of ***ETH_Fixed_Burst***
- ***uint32_t ETH_DMAInitTypeDef::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***
- ***uint32_t ETH_DMAInitTypeDef::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***
- ***uint32_t ETH_DMAInitTypeDef::DescriptorSkipLength*** Specifies the number of word to skip between two unchained descriptors (Ring mode) This parameter must be a number between Min_Data = 0 and Max_Data = 32
- ***uint32_t ETH_DMAInitTypeDef::DMAArbitration*** Selects the DMA Tx/Rx arbitration. This parameter can be a value of ***ETH_DMA_Arbitration***

15.1.4 ETH_DMADescTypeDef

ETH_DMADescTypeDef is defined in the stm32f1xx_hal_eth.h

Data Fields

- ***__IO uint32_t Status***
- ***uint32_t ControlBufferSize***
- ***uint32_t Buffer1Addr***
- ***uint32_t Buffer2NextDescAddr***

Field Documentation

- ***__IO uint32_t ETH_DMADescTypeDef::Status*** Status
- ***uint32_t ETH_DMADescTypeDef::ControlBufferSize*** Control and Buffer1, Buffer2 lengths
- ***uint32_t ETH_DMADescTypeDef::Buffer1Addr*** Buffer1 address pointer
- ***uint32_t ETH_DMADescTypeDef::Buffer2NextDescAddr*** Buffer2 or next descriptor address pointer

15.1.5 ETH_DMARxFrameInfos

ETH_DMARxFrameInfos is defined in the stm32f1xx_hal_eth.h

Data Fields

- ***ETH_DMADescTypeDef * FSRxDesc***
- ***ETH_DMADescTypeDef * LSRxDesc***
- ***uint32_t SegCount***
- ***uint32_t length***
- ***uint32_t buffer***

Field Documentation

- ***ETH_DMADescTypeDef* ETH_DMARxFrameInfos::FSRxDesc*** First Segment Rx Desc
- ***ETH_DMADescTypeDef* ETH_DMARxFrameInfos::LSRxDesc*** Last Segment Rx Desc
- ***uint32_t ETH_DMARxFrameInfos::SegCount*** Segment count
- ***uint32_t ETH_DMARxFrameInfos::length*** Frame length
- ***uint32_t ETH_DMARxFrameInfos::buffer*** Frame buffer

15.1.6 ETH_HandleTypeDef

ETH_HandleTypeDef is defined in the stm32f1xx_hal_eth.h

Data Fields

- ***ETH_TypeDef * Instance***
- ***ETH_InitTypeDef Init***
- ***uint32_t LinkStatus***
- ***ETH_DMADescTypeDef * RxDesc***
- ***ETH_DMADescTypeDef * TxDesc***
- ***ETH_DMARxFrameInfos RxFrameInfos***
- ***__IO HAL_ETH_StateTypeDef State***
- ***HAL_LockTypeDef Lock***

Field Documentation

- ***ETH_TypeDef* ETH_HandleTypeDef::Instance*** Register base address
- ***ETH_InitTypeDef ETH_HandleTypeDef::Init*** Ethernet Init Configuration
- ***uint32_t ETH_HandleTypeDef::LinkStatus*** Ethernet link status
- ***ETH_DMADescTypeDef* ETH_HandleTypeDef::RxDesc*** Rx descriptor to Get
- ***ETH_DMADescTypeDef* ETH_HandleTypeDef::TxDesc*** Tx descriptor to Set
- ***ETH_DMARxFrameInfos ETH_HandleTypeDef::RxFrameInfos*** last Rx frame infos
- ***__IO HAL_ETH_StateTypeDef ETH_HandleTypeDef::State*** ETH communication state
- ***HAL_LockTypeDef ETH_HandleTypeDef::Lock*** ETH Lock

15.2 ETH Firmware driver API description

The following section lists the various functions of the ETH library.

15.2.1 How to use this driver

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_MspInit() API:
   a. Enable the Ethernet interface clock using
      – __HAL_RCC_ETHMAC_CLK_ENABLE();

- __HAL_RCC_ETHMACTX_CLK_ENABLE();
- __HAL_RCC_ETHMACRX_CLK_ENABLE();

b. Initialize the related GPIO clocks
c. Configure Ethernet pin-out
d. Configure Ethernet NVIC interrupt (IT mode)

5. Initialize Ethernet DMA Descriptors in chain mode and point to allocated buffers:
   a. HAL_ETH_DMATxDescListInit(); for Transmission process
   b. HAL_ETH_DMARxDescListInit(); for Reception process
6. Enable MAC and DMA transmission and reception:
   a. HAL_ETH_Start();
7. Prepare ETH DMA TX Descriptors and give the hand to ETH DMA to transfer the frame to MAC TX FIFO:
   a. HAL_ETH_TransmitFrame();
8. Poll for a received frame in ETH RX DMA Descriptors and get received frame parameters
   a. HAL_ETH_GetReceivedFrame(); (should be called into an infinite loop)
9. Get a received frame when an ETH RX interrupt occurs:
   a. HAL_ETH_GetReceivedFrame_IT(); (called in IT mode only)
10. Communicate with external PHY device:
   a. Read a specific register from the PHY HAL_ETH_ReadPHYRegister();
   b. Write data to a specific RHY register: HAL_ETH_WritePHYRegister();
11. Configure the Ethernet MAC after ETH peripheral initialization HAL_ETH_ConfigMAC(); all MAC parameters should be filled.
12. Configure the Ethernet DMA after ETH peripheral initialization HAL_ETH_ConfigDMA(); all DMA parameters should be filled. The PTP protocol and the DMA descriptors ring mode are not supported in this driver

15.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the Ethernet peripheral
- De-initialize the Ethernet peripheral
- ***HAL_ETH_Init()***
- ***HAL_ETH_DeInit()***
- ***HAL_ETH_DMATxDescListInit()***
- ***HAL_ETH_DMARxDescListInit()***
- ***HAL_ETH_MspInit()***
- ***HAL_ETH_MspDeInit()***

15.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a frame HAL_ETH_TransmitFrame();
- Receive a frame HAL_ETH_GetReceivedFrame(); HAL_ETH_GetReceivedFrame_IT();
- Read from an External PHY register HAL_ETH_ReadPHYRegister();
- Write to an External PHY register HAL_ETH_WritePHYRegister();
- ***HAL_ETH_TransmitFrame()***
- ***HAL_ETH_GetReceivedFrame()***
- ***HAL_ETH_GetReceivedFrame_IT()***
- ***HAL_ETH_IRQHandler()***

- ***HAL_ETH_TxCpltCallback()***
- ***HAL_ETH_RxCpltCallback()***
- ***HAL_ETH_ErrorCallback()***
- ***HAL_ETH_ReadPHYRegister()***
- ***HAL_ETH_WritePHYRegister()***

15.2.4 Peripheral Control functions

This section provides functions allowing to:

- Enable MAC and DMA transmission and reception. HAL_ETH_Start();
- Disable MAC and DMA transmission and reception. HAL_ETH_Stop();
- Set the MAC configuration in runtime mode HAL_ETH_ConfigMAC();
- Set the DMA configuration in runtime mode HAL_ETH_ConfigDMA();
- ***HAL_ETH_Start()***
- ***HAL_ETH_Stop()***
- ***HAL_ETH_ConfigMAC()***
- ***HAL_ETH_ConfigDMA()***

15.2.5 Peripheral State functions

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

- Get the ETH handle state: HAL_ETH_GetState();
- ***HAL_ETH_GetState()***

15.2.6 HAL_ETH_Init

Function Name	**HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)**
Function Description	Initializes the Ethernet MAC and DMA according to default parameters.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• HAL status

15.2.7 HAL_ETH_DeInit

Function Name	**HAL_StatusTypeDef HAL_ETH_DeInit (ETH_HandleTypeDef * heth)**
Function Description	De-Initializes the ETH peripheral.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• HAL status

15.2.8 HAL_ETH_DMATxDescListInit

Function Name	**HAL_StatusTypeDef HAL_ETH_DMATxDescListInit (ETH_HandleTypeDef * heth, ETH_DMADescTypeDef * DMATxDescTab, uint8_t * TxBuff, uint32_t TxBuffCount)**
Function Description	Initializes the DMA Tx descriptors in chain mode.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module • **DMATxDescTab:** Pointer to the first Tx desc list • **TxBuff:** Pointer to the first TxBuffer list • **TxBuffCount:** Number of the used Tx desc in the list
Return values	• HAL status

15.2.9 HAL_ETH_DMARxDescListInit

Function Name	**HAL_StatusTypeDef HAL_ETH_DMARxDescListInit (ETH_HandleTypeDef * heth, ETH_DMADescTypeDef * DMARxDescTab, uint8_t * RxBuff, uint32_t RxBuffCount)**
Function Description	Initializes the DMA Rx descriptors in chain mode.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module • **DMARxDescTab:** Pointer to the first Rx desc list • **RxBuff:** Pointer to the first RxBuffer list • **RxBuffCount:** Number of the used Rx desc in the list
Return values	• HAL status

15.2.10 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

15.2.11 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

15.2.12 HAL_ETH_TransmitFrame

Function Name	**HAL_StatusTypeDef HAL_ETH_TransmitFrame (ETH_HandleTypeDef * heth, uint32_t FrameLength)**
Function Description	Sends an Ethernet frame.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module • **FrameLength:** Amount of data to be sent
Return values	• HAL status

15.2.13 HAL_ETH_GetReceivedFrame

Function Name	**HAL_StatusTypeDef HAL_ETH_GetReceivedFrame (ETH_HandleTypeDef * heth)**
Function Description	Checks for received frames.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• HAL status

15.2.14 HAL_ETH_GetReceivedFrame_IT

Function Name	**HAL_StatusTypeDef HAL_ETH_GetReceivedFrame_IT (ETH_HandleTypeDef * heth)**
Function Description	Gets the Received frame in interrupt mode.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• HAL status

15.2.15 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

15.2.16 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

15.2.17 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

15.2.18 HAL_ETH_ErrorCallback

Function Name	**void HAL_ETH_ErrorCallback (ETH_HandleTypeDef * heth)**
Function Description	Ethernet transfer error callbacks.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• None

15.2.19 HAL_ETH_ReadPHYRegister

Function Name	**HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t * RegValue)**
Function Description	Reads a PHY register.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module • **PHYReg:** PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY_BCR: Transceiver Basic Control Register, PHY_BSR: Transceiver Basic Status Register. More PHY register could be read depending on the used PHY • **RegValue:** PHY register value
Return values	• HAL status

15.2.20 HAL_ETH_WritePHYRegister

Function Name	**HAL_StatusTypeDef HAL_ETH_WritePHYRegister (ETH_HandleTypeDef * heth, uint16_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 • **PHYReg:** PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY_BCR: Transceiver Control Register. More PHY register could be written depending on the used PHY • **RegValue:** the value to write
Return values	• HAL status

15.2.21 HAL_ETH_Start

Function Name	**HAL_StatusTypeDef HAL_ETH_Start (ETH_HandleTypeDef * heth)**
Function Description	Enables 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

15.2.22 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

15.2.23 HAL_ETH_ConfigMAC

Function Name	**HAL_StatusTypeDef HAL_ETH_ConfigMAC (ETH_HandleTypeDef * heth, ETH_MACInitTypeDef * macconf)**
Function Description	Set ETH MAC Configuration.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module • **macconf:** MAC Configuration structure
Return values	• HAL status

15.2.24 HAL_ETH_ConfigDMA

Function Name	**HAL_StatusTypeDef HAL_ETH_ConfigDMA (ETH_HandleTypeDef * heth, ETH_DMAInitTypeDef * dmaconf)**
Function Description	Sets ETH DMA Configuration.

Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module • **dmaconf:** DMA Configuration structure
Return values	• HAL status

15.2.25 HAL_ETH_GetState

Function Name	**HAL_ETH_StateTypeDef HAL_ETH_GetState (ETH_HandleTypeDef * heth)**
Function Description	Return the ETH HAL state.
Parameters	• **heth:** pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
Return values	• HAL state

15.3 ETH Firmware driver defines

The following section lists the various define and macros of the module.

15.3.1 ETH

ETH

ETH Address Aligned Beats

ETH_ADDRESSALIGNEDBEATS_ENABLE

ETH_ADDRESSALIGNEDBEATS_DISABLE

ETH Automatic Pad CRC Strip

ETH_AUTOMATICPADCRCSTRIP_ENABLE

ETH_AUTOMATICPADCRCSTRIP_DISABLE

ETH AutoNegotiation

ETH_AUTONEGOTIATION_ENABLE

ETH_AUTONEGOTIATION_DISABLE

ETH Back Off Limit

ETH_BACKOFFLIMIT_10

ETH_BACKOFFLIMIT_8

ETH_BACKOFFLIMIT_4

ETH_BACKOFFLIMIT_1

ETH Broadcast Frames Reception

ETH_BROADCASTFRAMESRECEPTION_ENABLE

ETH_BROADCASTFRAMESRECEPTION_DISABLE

ETH Buffers setting

ETH_MAX_PACKET_SIZE	ETH_HEADER + ETH_EXTRA + ETH_VLAN_TAG + ETH_MAX_ETH_PAYLOAD + ETH_CRC

ETH_HEADER	6 byte Dest addr, 6 byte Src addr, 2 byte length/type
ETH_CRC	Ethernet CRC
ETH_EXTRA	Extra bytes in some cases
ETH_VLAN_TAG	optional 802.1q VLAN Tag
ETH_MIN_ETH_PAYLOAD	Minimum Ethernet payload size
ETH_MAX_ETH_PAYLOAD	Maximum Ethernet payload size
ETH_JUMBO_FRAME_PAYLOAD	Jumbo frame payload size
ETH_RX_BUF_SIZE	
ETH_RXBUFNB	
ETH_TX_BUF_SIZE	
ETH_TXBUFNB	

ETH Carrier Sense

ETH_CARRIERSENCE_ENABLE

ETH_CARRIERSENCE_DISABLE

ETH Checksum Mode

ETH_CHECKSUM_BY_HARDWARE

ETH_CHECKSUM_BY_SOFTWARE

ETH Checksum Offload

ETH_CHECKSUMOFFLAOD_ENABLE

ETH_CHECKSUMOFFLAOD_DISABLE

ETH Deferral Check

ETH_DEFFERRALCHECK_ENABLE

ETH_DEFFERRALCHECK_DISABLE

ETH Destination Addr Filter

ETH_DESTINATIONADDRFILTER_NORMAL

ETH_DESTINATIONADDRFILTER_INVERSE

ETH DMA Arbitration

ETH_DMAARBITRATION_ROUNDROBIN_RXTX_1_1

ETH_DMAARBITRATION_ROUNDROBIN_RXTX_2_1

ETH_DMAARBITRATION_ROUNDROBIN_RXTX_3_1

ETH_DMAARBITRATION_ROUNDROBIN_RXTX_4_1

ETH_DMAARBITRATION_RXPRIORTX

ETH DMA Flags

ETH_DMA_FLAG_TST	Time-stamp trigger interrupt (on DMA)
ETH_DMA_FLAG_PMT	PMT interrupt (on DMA)
ETH_DMA_FLAG_MMC	MMC interrupt (on DMA)

ETH_DMA_FLAG_DATATRANSFERERROR	Error bits 0-Rx DMA, 1-Tx DMA
ETH_DMA_FLAG_READWRITEERROR	Error bits 0-write trnsf, 1-read transfr
ETH_DMA_FLAG_ACCESSERROR	Error bits 0-data buffer, 1-desc. access
ETH_DMA_FLAG_NIS	Normal interrupt summary flag
ETH_DMA_FLAG_AIS	Abnormal interrupt summary flag
ETH_DMA_FLAG_ER	Early receive flag
ETH_DMA_FLAG_FBE	Fatal bus error flag
ETH_DMA_FLAG_ET	Early transmit flag
ETH_DMA_FLAG_RWT	Receive watchdog timeout flag
ETH_DMA_FLAG_RPS	Receive process stopped flag
ETH_DMA_FLAG_RBU	Receive buffer unavailable flag
ETH_DMA_FLAG_R	Receive flag
ETH_DMA_FLAG_TU	Underflow flag
ETH_DMA_FLAG_RO	Overflow flag
ETH_DMA_FLAG_TJT	Transmit jabber timeout flag
ETH_DMA_FLAG_TBU	Transmit buffer unavailable flag
ETH_DMA_FLAG_TPS	Transmit process stopped flag
ETH_DMA_FLAG_T	Transmit flag

ETH DMA Interrupts

ETH_DMA_IT_TST	Time-stamp trigger interrupt (on DMA)
ETH_DMA_IT_PMT	PMT interrupt (on DMA)
ETH_DMA_IT_MMC	MMC interrupt (on DMA)
ETH_DMA_IT_NIS	Normal interrupt summary
ETH_DMA_IT_AIS	Abnormal interrupt summary
ETH_DMA_IT_ER	Early receive interrupt
ETH_DMA_IT_FBE	Fatal bus error interrupt
ETH_DMA_IT_ET	Early transmit interrupt
ETH_DMA_IT_RWT	Receive watchdog timeout interrupt
ETH_DMA_IT_RPS	Receive process stopped interrupt
ETH_DMA_IT_RBU	Receive buffer unavailable interrupt
ETH_DMA_IT_R	Receive interrupt
ETH_DMA_IT_TU	Underflow interrupt
ETH_DMA_IT_RO	Overflow interrupt
ETH_DMA_IT_TJT	Transmit jabber timeout interrupt
ETH_DMA_IT_TBU	Transmit buffer unavailable interrupt
ETH_DMA_IT_TPS	Transmit process stopped interrupt

ETH_DMA_IT_T	Transmit interrupt

ETH DMA overflow

ETH_DMA_OVERFLOW_RXFIFOCOUNTER	Overflow bit for FIFO overflow counter
ETH_DMA_OVERFLOW_MISSEDFRAMECOUNTER	Overflow bit for missed frame counter

ETH DMA receive process state

ETH_DMA_RECEIVEPROCESS_STOPPED	Stopped - Reset or Stop Rx Command issued
ETH_DMA_RECEIVEPROCESS_FETCHING	Running - fetching the Rx descriptor
ETH_DMA_RECEIVEPROCESS_WAITING	Running - waiting for packet
ETH_DMA_RECEIVEPROCESS_SUSPENDED	Suspended - Rx Descriptor unavailable
ETH_DMA_RECEIVEPROCESS_CLOSING	Running - closing descriptor
ETH_DMA_RECEIVEPROCESS_QUEUING	Running - queuing the receive frame into host memory

ETH DMA RX Descriptor

ETH_DMARXDESC_OWN	OWN bit: descriptor is owned by DMA engine
ETH_DMARXDESC_AFM	DA Filter Fail for the rx frame
ETH_DMARXDESC_FL	Receive descriptor frame length
ETH_DMARXDESC_ES	Error summary: OR of the following bits: DE || OE || IPC || LC || RWT || RE || CE
ETH_DMARXDESC_DE	Descriptor error: no more descriptors for receive frame
ETH_DMARXDESC_SAF	SA Filter Fail for the received frame
ETH_DMARXDESC_LE	Frame size not matching with length field
ETH_DMARXDESC_OE	Overflow Error: Frame was damaged due to buffer overflow
ETH_DMARXDESC_VLAN	VLAN Tag: received frame is a VLAN frame
ETH_DMARXDESC_FS	First descriptor of the frame
ETH_DMARXDESC_LS	Last descriptor of the frame
ETH_DMARXDESC_IPV4HCE	IPC Checksum Error: Rx Ipv4 header checksum error
ETH_DMARXDESC_LC	Late collision occurred during reception
ETH_DMARXDESC_FT	Frame type - Ethernet, otherwise 802.3
ETH_DMARXDESC_RWT	Receive Watchdog Timeout: watchdog timer expired during reception
ETH_DMARXDESC_RE	Receive error: error reported by MII interface
ETH_DMARXDESC_DBE	Dribble bit error: frame contains non int multiple of 8 bits
ETH_DMARXDESC_CE	CRC error
ETH_DMARXDESC_MAMPCE	Rx MAC Address/Payload Checksum Error: Rx MAC address matched/ Rx Payload Checksum Error

ETH_DMARXDESC_DIC	Disable Interrupt on Completion
ETH_DMARXDESC_RBS2	Receive Buffer2 Size
ETH_DMARXDESC_RER	Receive End of Ring
ETH_DMARXDESC_RCH	Second Address Chained
ETH_DMARXDESC_RBS1	Receive Buffer1 Size
ETH_DMARXDESC_B1AP	Buffer1 Address Pointer
ETH_DMARXDESC_B2AP	Buffer2 Address Pointer

ETH DMA Rx Descriptor Buffers

ETH_DMARXDESC_BUFFER1	DMA Rx Desc Buffer1
ETH_DMARXDESC_BUFFER2	DMA Rx Desc Buffer2

ETH DMA transmit process state

ETH_DMA_TRANSMITPROCESS_STOPPED	Stopped - Reset or Stop Tx Command issued
ETH_DMA_TRANSMITPROCESS_FETCHING	Running - fetching the Tx descriptor
ETH_DMA_TRANSMITPROCESS_WAITING	Running - waiting for status
ETH_DMA_TRANSMITPROCESS_READING	Running - reading the data from host memory
ETH_DMA_TRANSMITPROCESS_SUSPENDED	Suspended - Tx Descriptor unavailable
ETH_DMA_TRANSMITPROCESS_CLOSING	Running - closing Rx descriptor

ETH DMA TX Descriptor

ETH_DMATXDESC_OWN	OWN bit: descriptor is owned by DMA engine
ETH_DMATXDESC_IC	Interrupt on Completion
ETH_DMATXDESC_LS	Last Segment
ETH_DMATXDESC_FS	First Segment
ETH_DMATXDESC_DC	Disable CRC
ETH_DMATXDESC_DP	Disable Padding
ETH_DMATXDESC_TTSE	Transmit Time Stamp Enable
ETH_DMATXDESC_CIC	Checksum Insertion Control: 4 cases
ETH_DMATXDESC_CIC_BYPASS	Do Nothing: Checksum Engine is bypassed
ETH_DMATXDESC_CIC_IPV4HEADER	IPV4 header Checksum Insertion
ETH_DMATXDESC_CIC_TCPUDPICMP_SEGMENT	TCP/UDP/ICMP Checksum Insertion calculated over segment only
ETH_DMATXDESC_CIC_TCPUDPICMP_FULL	TCP/UDP/ICMP Checksum Insertion fully calculated

ETH_DMATXDESC_TER	Transmit End of Ring
ETH_DMATXDESC_TCH	Second Address Chained
ETH_DMATXDESC_TTSS	Tx Time Stamp Status
ETH_DMATXDESC_IHE	IP Header Error
ETH_DMATXDESC_ES	Error summary: OR of the following bits: UE || ED || EC || LCO || NC || LCA || FF || JT
ETH_DMATXDESC_JT	Jabber Timeout
ETH_DMATXDESC_FF	Frame Flushed: DMA/MTL flushed the frame due to SW flush
ETH_DMATXDESC_PCE	Payload Checksum Error
ETH_DMATXDESC_LCA	Loss of Carrier: carrier lost during transmission
ETH_DMATXDESC_NC	No Carrier: no carrier signal from the transceiver
ETH_DMATXDESC_LCO	Late Collision: transmission aborted due to collision
ETH_DMATXDESC_EC	Excessive Collision: transmission aborted after 16 collisions
ETH_DMATXDESC_VF	VLAN Frame
ETH_DMATXDESC_CC	Collision Count
ETH_DMATXDESC_ED	Excessive Deferral
ETH_DMATXDESC_UF	Underflow Error: late data arrival from the memory
ETH_DMATXDESC_DB	Deferred Bit
ETH_DMATXDESC_TBS2	Transmit Buffer2 Size
ETH_DMATXDESC_TBS1	Transmit Buffer1 Size
ETH_DMATXDESC_B1AP	Buffer1 Address Pointer
ETH_DMATXDESC_B2AP	Buffer2 Address Pointer

ETH DMA Tx Descriptor Checksum Insertion Control

ETH_DMATXDESC_CHECKSUMBYPASS	Checksum engine bypass
ETH_DMATXDESC_CHECKSUMIPV4HEADER	IPv4 header checksum insertion
ETH_DMATXDESC_CHECKSUMTCPUDPICMPSEGMENT	TCP/UDP/ICMP checksum insertion. Pseudo header checksum is assumed to be present
ETH_DMATXDESC_CHECKSUMTCPUDPICMPFULL	TCP/UDP/ICMP checksum fully in hardware including pseudo header

ETH DMA Tx Descriptor Segment

ETH_DMATXDESC_LASTSEGMENTS	Last Segment
ETH_DMATXDESC_FIRSTSEGMENT	First Segment

ETH Drop TCP IP Checksum Error Frame

ETH_DROPTCPIPCHECKSUMERRORFRAME_ENABLE

ETH_DROPTCPIPCHECKSUMERRORFRAME_DISABLE

ETH Duplex Mode

ETH_MODE_FULLDUPLEX

ETH_MODE_HALFDUPLEX

ETH Exported Macros

__HAL_ETH_RESET_HANDLE_STATE

Description:

- Reset ETH handle state.

Parameters:

- __HANDLE__: specifies the ETH handle.

Return value:

- None:

__HAL_ETH_DMATXDESC_GET_FLAG

Description:

- Checks whether the specified ETHERNET DMA Tx Desc flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag of TDES0 to check .

Return value:

- the: ETH_DMATxDescFlag (SET or RESET).

__HAL_ETH_DMARXDESC_GET_FLAG

Description:

- Checks whether the specified ETHERNET DMA Rx Desc flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag of RDES0 to check.

Return value:

- the: ETH_DMATxDescFlag (SET or RESET).

__HAL_ETH_DMARXDESC_ENABLE_IT

Description:

- Enables the specified DMA Rx Desc

	receive interrupt. **Parameters:** • __HANDLE__: ETH Handle **Return value:** • None:
__HAL_ETH_DMARXDESC_DISABLE_IT	**Description:** • Disables the specified DMA Rx Desc receive interrupt. **Parameters:** • __HANDLE__: ETH Handle **Return value:** • None:
__HAL_ETH_DMARXDESC_SET_OWN_BIT	**Description:** • Set the specified DMA Rx Desc Own bit. **Parameters:** • __HANDLE__: ETH Handle **Return value:** • None:
__HAL_ETH_DMATXDESC_GET_COLLISION_COUNT	**Description:** • Returns the specified ETHERNET DMA Tx Desc collision count. **Parameters:** • __HANDLE__: ETH Handle **Return value:** • The: Transmit descriptor collision counter value.
__HAL_ETH_DMATXDESC_SET_OWN_BIT	**Description:** • Set the specified DMA Tx Desc Own bit. **Parameters:** • __HANDLE__: ETH Handle **Return value:** • None:
__HAL_ETH_DMATXDESC_ENABLE_IT	**Description:** • Enables the specified DMA Tx Desc Transmit interrupt. **Parameters:** • __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_DMATXDESC_DISABLE_IT

Description:

- Disables the specified DMA Tx Desc Transmit interrupt.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_DMATXDESC_CHECKSUM_INSERTION

Description:

- Selects the specified ETHERNET DMA Tx Desc Checksum Insertion.

Parameters:

- __HANDLE__: ETH Handle
- __CHECKSUM__: specifies is the DMA Tx desc checksum insertion. This parameter can be one of the following values:
  - ETH_DMATXDESC_CHECKSUMBYPASS : Checksum bypass
  - ETH_DMATXDESC_CHECKSUMIPV4HEADER : IPv4 header checksum
  - ETH_DMATXDESC_CHECKSUMTCPUDPICMPSEGMENT : TCP/UDP/ICMP checksum. Pseudo header checksum is assumed to be present
  - ETH_DMATXDESC_CHECKSUMTCPUDPICMPFULL : TCP/UDP/ICMP checksum fully in hardware including pseudo header

Return value:

- None:

__HAL_ETH_DMATXDESC_CRC_ENABLE

Description:

- Enables the DMA Tx Desc CRC.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_DMATXDESC_CRC_DISABLE

Description:

- Disables the DMA Tx Desc CRC.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_DMATXDESC_SHORT_FRAME_PADDING_ENABLE

Description:

- Enables the DMA Tx Desc padding for frame shorter than 64 bytes.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_DMATXDESC_SHORT_FRAME_PADDING_DISABLE

Description:

- Disables the DMA Tx Desc padding for frame shorter than 64 bytes.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__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 or disabled. This parameter can be any combination of the following values:
  – ETH_MAC_IT_TST : Time stamp trigger interrupt
  – ETH_MAC_IT_PMT : PMT interrupt

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 or disabled. This parameter can be any combination of the following values:

- ETH_MAC_IT_TST : Time stamp trigger interrupt
- ETH_MAC_IT_PMT : PMT interrupt

Return value:

- None:

__HAL_ETH_INITIATE_PAUSE_CONTROL_FRAME

Description:

- Initiate a Pause Control Frame (Full-duplex only).

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_GET_FLOW_CONTROL_BUSY_STATUS

Description:

- Checks whether the ETHERNET flow control busy bit is set or not.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- The: new state of flow control busy status bit (SET or RESET).

__HAL_ETH_BACK_PRESSURE_ACTIVATION_ENABLE

Description:

- Enables the MAC Back Pressure operation activation (Half-duplex only).

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_BACK_PRESSURE_ACTIVATION_DISABLE

Description:

- Disables the MAC BackPressure operation activation (Half-duplex only).

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_MAC_GET_FLAG

Description:

- Checks whether the specified ETHERNET MAC flag is set or not.

Parameters:

- __HANDLE__: ETH Handle

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - ETH_MAC_FLAG_TST : Time stamp trigger flag
  - ETH_MAC_FLAG_MMCT : MMC transmit flag
  - ETH_MAC_FLAG_MMCR : MMC receive flag
  - ETH_MAC_FLAG_MMC : MMC flag
  - ETH_MAC_FLAG_PMT : PMT flag

Return value:

- The: state of ETHERNET MAC flag.

__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_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: new state of ETH_DMA_FLAG (SET or RESET).

__HAL_ETH_DMA_CLEAR_FLAG

Description:

- Checks whether the specified ETHERNET DMA flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __FLAG__: specifies the flag to clear.

Return value:

- The: new state of ETH_DMA_FLAG (SET or RESET).

__HAL_ETH_GET_DMA_OVERFLOW_STATUS

Description:

- Checks whether the specified ETHERNET DMA overflow flag is set or not.

Parameters:

- __HANDLE__: ETH Handle
- __OVERFLOW__: specifies the DMA overflow flag to check. This parameter can be one of the following values:
  – ETH_DMA_OVERFLOW_RXFIFOCOUNTER : Overflow for FIFO Overflows Counter
  – ETH_DMA_OVERFLOW_MISSEDFRAMECOUNTER : Overflow for Buffer Unavailable Missed Frame Counter

Return value:

- The: state of ETHERNET DMA overflow Flag (SET or RESET).

__HAL_ETH_SET_RECEIVE_WATCHDOG_TIMER

Description:

- Set the DMA Receive status watchdog timer register value.

Parameters:

- __HANDLE__: ETH Handle
- __VALUE__: DMA Receive status

watchdog timer register value

Return value:

- None:

__HAL_ETH_GLOBAL_UNICAST_WAKEUP_ENABLE

Description:

- Enables any unicast packet filtered by the MAC address recognition to be a wake-up frame.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_GLOBAL_UNICAST_WAKEUP_DISABLE

Description:

- Disables any unicast packet filtered by the MAC address recognition to be a wake-up frame.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_WAKEUP_FRAME_DETECTION_ENABLE

Description:

- Enables the MAC Wake-Up Frame Detection.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_WAKEUP_FRAME_DETECTION_DISABLE

Description:

- Disables the MAC Wake-Up Frame Detection.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_MAGIC_PACKET_DETECTION_ENABLE

Description:

- Enables the MAC Magic Packet Detection.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_MAGIC_PACKET_DETECTION_DISABLE

Description:

- Disables the MAC Magic Packet Detection.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_POWER_DOWN_ENABLE

Description:

- Enables the MAC Power Down.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_POWER_DOWN_DISABLE

Description:

- Disables the MAC Power Down.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None:

__HAL_ETH_GET_PMT_FLAG_STATUS

Description:

- Checks whether the specified ETHERNET PMT flag is set or not.

Parameters:

- __HANDLE__: ETH Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - ETH_PMT_FLAG_WUFFRPR : Wake-Up Frame Filter Register Pointer Reset
  - ETH_PMT_FLAG_WUFR : Wake-Up Frame Received
  - ETH_PMT_FLAG_MPR : Magic Packet Received

Return value:

- The: new state of ETHERNET PMT Flag (SET or RESET).

__HAL_ETH_MMC_COUNTER_FULL_PR

Description:

ESET	• Preset and Initialize the MMC counters to almost-full value: 0xFFFF_FFF0 (full - 16) **Parameters:** • __HANDLE__: ETH Handle. **Return value:** • None:
__HAL_ETH_MMC_COUNTER_HALF_PRESET	**Description:** • Preset and Initialize the MMC counters to almost-half value: 0x7FFF_FFF0 (half - 16) **Parameters:** • __HANDLE__: ETH Handle. **Return value:** • None:
__HAL_ETH_MMC_COUNTER_FREEZE_ENABLE	**Description:** • Enables the MMC Counter Freeze. **Parameters:** • __HANDLE__: ETH Handle. **Return value:** • None:
__HAL_ETH_MMC_COUNTER_FREEZE_DISABLE	**Description:** • Disables the MMC Counter Freeze. **Parameters:** • __HANDLE__: ETH Handle. **Return value:** • None:
__HAL_ETH_ETH_MMC_RESET_ONREAD_ENABLE	**Description:** • Enables the MMC Reset On Read. **Parameters:** • __HANDLE__: ETH Handle. **Return value:** • None:
__HAL_ETH_ETH_MMC_RESET_ONREAD_DISABLE	**Description:** • Disables the MMC Reset On Read. **Parameters:** • __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_ETH_MMC_COUNTER_ROLLOVER_ENABLE

Description:

- Enables the MMC Counter Stop Rollover.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_ETH_MMC_COUNTER_ROLLOVER_DISABLE

Description:

- Disables the MMC Counter Stop Rollover.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_MMC_COUNTERS_RESET

Description:

- Resets the MMC Counters.

Parameters:

- __HANDLE__: ETH Handle.

Return value:

- None:

__HAL_ETH_MMC_RX_IT_ENABLE

Description:

- Enables the specified ETHERNET MMC Rx interrupts.

Parameters:

- __HANDLE__: ETH Handle.
- __INTERRUPT__: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH_MMC_IT_RGUF : When Rx good unicast frames counter reaches half the maximum value
  - ETH_MMC_IT_RFAE : When Rx alignment error counter reaches half the maximum value
  - ETH_MMC_IT_RFCE : When Rx crc error counter reaches half the maximum value

Return value:

- None:

__HAL_ETH_MMC_RX_IT_DISABLE

Description:

- Disables the specified ETHERNET MMC Rx interrupts.

Parameters:

- __HANDLE__: ETH Handle.
- __INTERRUPT__: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH_MMC_IT_RGUF : When Rx good unicast frames counter reaches half the maximum value
  - ETH_MMC_IT_RFAE : When Rx alignment error counter reaches half the maximum value
  - ETH_MMC_IT_RFCE : When Rx crc error counter reaches half the maximum value

Return value:

- None:

__HAL_ETH_MMC_TX_IT_ENABLE

Description:

- Enables the specified ETHERNET MMC Tx interrupts.

Parameters:

- __HANDLE__: ETH Handle.
- __INTERRUPT__: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH_MMC_IT_TGF : When Tx good frame counter reaches half the maximum value
  - ETH_MMC_IT_TGFMSC: When Tx good multi col counter reaches half the maximum value
  - ETH_MMC_IT_TGFSC : When Tx good single col counter reaches half the maximum value

Return value:

- None:

__HAL_ETH_MMC_TX_IT_DISABLE

Description:

- Disables the specified ETHERNET MMC Tx interrupts.

Parameters:

- __HANDLE__: ETH Handle.

- __INTERRUPT__: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH_MMC_IT_TGF : When Tx good frame counter reaches half the maximum value
  - ETH_MMC_IT_TGFMSC: When Tx good multi col counter reaches half the maximum value
  - ETH_MMC_IT_TGFSC : When Tx good single col counter reaches half the maximum value

Return value:

- None:

__HAL_ETH_WAKEUP_EXTI_ENABLE_IT

Description:

- Enables the ETH External interrupt line.

Return value:

- None:

__HAL_ETH_WAKEUP_EXTI_DISABLE_IT

Description:

- Disables the ETH External interrupt line.

Return value:

- None:

__HAL_ETH_WAKEUP_EXTI_ENABLE_EVENT

Description:

- Enable event on ETH External event line.

Return value:

- None.:

__HAL_ETH_WAKEUP_EXTI_DISABLE_EVENT

Description:

- Disable event on ETH External event line.

Return value:

- None.:

__HAL_ETH_WAKEUP_EXTI_GET_FLAG

Description:

- Get flag of the ETH External interrupt line.

Return value:

- None:

__HAL_ETH_WAKEUP_EXTI_CLEAR_FLAG

Description:

- Clear flag of the ETH External interrupt line.

	Return value: • None:
__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_EDGE_TRIGGER	**Description:** • Enables rising edge trigger to the ETH External interrupt line. **Return value:** • None:
__HAL_ETH_WAKEUP_EXTI_DISABLE_RISING_EDGE_TRIGGER	**Description:** • Disables the rising edge trigger to the ETH External interrupt line. **Return value:** • None:
__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLING_EDGE_TRIGGER	**Description:** • Enables falling edge trigger to the ETH External interrupt line. **Return value:** • None:
__HAL_ETH_WAKEUP_EXTI_DISABLE_FALLING_EDGE_TRIGGER	**Description:** • Disables falling edge trigger to the ETH External interrupt line. **Return value:** • None:
__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLINGRISING_TRIGGER	**Description:** • Enables rising/falling edge trigger to the ETH External interrupt line. **Return value:** • None:
__HAL_ETH_WAKEUP_EXTI_DISABLE_FALLINGRISING_TRIGGER	**Description:** • Disables rising/falling edge trigger to the ETH External interrupt line. **Return value:** • None:
__HAL_ETH_WAKEUP_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None.:

ETH EXTI LINE WAKEUP

ETH_EXTI_LINE_WAKEUP	External interrupt line 19 Connected to the ETH EXTI Line

ETH Fixed Burst

ETH_FIXEDBURST_ENABLE

ETH_FIXEDBURST_DISABLE

ETH Flush Received Frame

ETH_FLUSHRECEIVEDFRAME_ENABLE

ETH_FLUSHRECEIVEDFRAME_DISABLE

ETH Forward Error Frames

ETH_FORWARDERRORFRAMES_ENABLE

ETH_FORWARDERRORFRAMES_DISABLE

ETH Forward Undersized Good Frames

ETH_FORWARDUNDERSIZEDGOODFRAMES_ENABLE

ETH_FORWARDUNDERSIZEDGOODFRAMES_DISABLE

ETH Inter Frame Gap

ETH_INTERFRAMEGAP_96BIT	minimum IFG between frames during transmission is 96Bit
ETH_INTERFRAMEGAP_88BIT	minimum IFG between frames during transmission is 88Bit
ETH_INTERFRAMEGAP_80BIT	minimum IFG between frames during transmission is 80Bit
ETH_INTERFRAMEGAP_72BIT	minimum IFG between frames during transmission is 72Bit
ETH_INTERFRAMEGAP_64BIT	minimum IFG between frames during transmission is 64Bit
ETH_INTERFRAMEGAP_56BIT	minimum IFG between frames during transmission is 56Bit
ETH_INTERFRAMEGAP_48BIT	minimum IFG between frames during transmission is 48Bit
ETH_INTERFRAMEGAP_40BIT	minimum IFG between frames during transmission is 40Bit

ETH Jabber

ETH_JABBER_ENABLE

ETH_JABBER_DISABLE

ETH Loop Back Mode

ETH_LOOPBACKMODE_ENABLE

ETH_LOOPBACKMODE_DISABLE

ETH MAC addresses

ETH_MAC_ADDRESS0

ETH_MAC_ADDRESS1

ETH_MAC_ADDRESS2

ETH_MAC_ADDRESS3

ETH_MAC Addresses Filter Mask Bytes

ETH_MAC_ADDRESSMASK_BYTE6	Mask MAC Address high reg bits [15:8]
ETH_MAC_ADDRESSMASK_BYTE5	Mask MAC Address high reg bits [7:0]
ETH_MAC_ADDRESSMASK_BYTE4	Mask MAC Address low reg bits [31:24]
ETH_MAC_ADDRESSMASK_BYTE3	Mask MAC Address low reg bits [23:16]
ETH_MAC_ADDRESSMASK_BYTE2	Mask MAC Address low reg bits [15:8]
ETH_MAC_ADDRESSMASK_BYTE1	Mask MAC Address low reg bits [70]

ETH MAC Addresses Filter SA DA

ETH_MAC_ADDRESSFILTER_SA

ETH_MAC_ADDRESSFILTER_DA

ETH MAC Debug Flags

ETH_MAC_TXFIFO_FULL

ETH_MAC_TXFIFONOT_EMPTY

ETH_MAC_TXFIFO_WRITE_ACTIVE

ETH_MAC_TXFIFO_IDLE

ETH_MAC_TXFIFO_READ

ETH_MAC_TXFIFO_WAITING

ETH_MAC_TXFIFO_WRITING

ETH_MAC_TRANSMISSION_PAUSE

ETH_MAC_TRANSMITFRAMECONTROLLER_IDLE

ETH_MAC_TRANSMITFRAMECONTROLLER_WAITING

ETH_MAC_TRANSMITFRAMECONTROLLER_GENRATING_PCF

ETH_MAC_TRANSMITFRAMECONTROLLER_TRANSFERRING

ETH_MAC_MII_TRANSMIT_ACTIVE

ETH_MAC_RXFIFO_EMPTY

ETH_MAC_RXFIFO_BELOW_THRESHOLD

ETH_MAC_RXFIFO_ABOVE_THRESHOLD

ETH_MAC_RXFIFO_FULL

ETH_MAC_READCONTROLLER_IDLE

ETH_MAC_READCONTROLLER_READING_DATA

ETH_MAC_READCONTROLLER_READING_STATUS

ETH_MAC_READCONTROLLER_

ETH_MAC_RXFIFO_WRITE_ACTIVE

ETH_MAC_SMALL_FIFO_NOTACTIVE

ETH_MAC_SMALL_FIFO_READ_ACTIVE

ETH_MAC_SMALL_FIFO_WRITE_ACTIVE

ETH_MAC_SMALL_FIFO_RW_ACTIVE

ETH_MAC_MII_RECEIVE_PROTOCOL_ACTIVE

ETH MAC Flags

ETH_MAC_FLAG_TST	Time stamp trigger flag (on MAC)
ETH_MAC_FLAG_MMCT	MMC transmit flag
ETH_MAC_FLAG_MMCR	MMC receive flag
ETH_MAC_FLAG_MMC	MMC flag (on MAC)
ETH_MAC_FLAG_PMT	PMT flag (on MAC)

ETH MAC Interrupts

ETH_MAC_IT_TST	Time stamp trigger interrupt (on MAC)
ETH_MAC_IT_MMCT	MMC transmit interrupt
ETH_MAC_IT_MMCR	MMC receive interrupt
ETH_MAC_IT_MMC	MMC interrupt (on MAC)
ETH_MAC_IT_PMT	PMT interrupt (on MAC)

ETH Media Interface

ETH_MEDIA_INTERFACE_MII

ETH_MEDIA_INTERFACE_RMII

ETH MMC Rx Interrupts

ETH_MMC_IT_RGUF	When Rx good unicast frames counter reaches half the maximum value
ETH_MMC_IT_RFAE	When Rx alignment error counter reaches half the maximum value
ETH_MMC_IT_RFCE	When Rx crc error counter reaches half the maximum value

ETH MMC Tx Interrupts

ETH_MMC_IT_TGF	When Tx good frame counter reaches half the maximum value
ETH_MMC_IT_TGFMSC	When Tx good multi col counter reaches half the maximum value
ETH_MMC_IT_TGFSC	When Tx good single col counter reaches half the maximum value

ETH Multicast Frames Filter

ETH_MULTICASTFRAMESFILTER_PERFECTHASHTABLE

ETH_MULTICASTFRAMESFILTER_HASHTABLE

ETH_MULTICASTFRAMESFILTER_PERFECT

ETH_MULTICASTFRAMESFILTER_NONE

ETH Pass Control Frames

ETH_PASSCONTROLFRAMES_BLOCKALL	MAC filters all control frames from

	reaching the application
ETH_PASSCONTROLFRAMES_FORWARDALL	MAC forwards all control frames to application even if they fail the Address Filter
ETH_PASSCONTROLFRAMES_FORWARDPASSEDADDRFILTER	MAC forwards control frames that pass the Address Filter.

ETH Pause Low Threshold

ETH_PAUSELOWTHRESHOLD_MINUS4	Pause time minus 4 slot times
ETH_PAUSELOWTHRESHOLD_MINUS28	Pause time minus 28 slot times
ETH_PAUSELOWTHRESHOLD_MINUS144	Pause time minus 144 slot times
ETH_PAUSELOWTHRESHOLD_MINUS256	Pause time minus 256 slot times

ETH PMT Flags

ETH_PMT_FLAG_WUFFRPR	Wake-Up Frame Filter Register Pointer Reset
ETH_PMT_FLAG_WUFR	Wake-Up Frame Received
ETH_PMT_FLAG_MPR	Magic Packet Received

ETH Private Constants

LINKED_STATE_TIMEOUT_VALUE

AUTONEGO_COMPLETED_TIMEOUT_VALUE

ETH_REG_WRITE_DELAY

ETH_SUCCESS

ETH_ERROR

ETH_DMATXDESC_COLLISION_COUNTSHIFT

ETH_DMATXDESC_BUFFER2_SIZESHIFT

ETH_DMARXDESC_FRAME_LENGTHSHIFT

ETH_DMARXDESC_BUFFER2_SIZESHIFT

ETH_DMARXDESC_FRAMELENGTHSHIFT

ETH_MAC_ADDR_HBASE

ETH_MAC_ADDR_LBASE

ETH_MACMIIAR_CR_MASK

ETH_MACCR_CLEAR_MASK

ETH_MACFCR_CLEAR_MASK

ETH_DMAOMR_CLEAR_MASK

ETH_WAKEUP_REGISTER_LENGTH

ETH_DMA_RX_OVERFLOW_MISSEDFRAMES_COUNTERSHIFT

ETH_Private_Macros

IS_ETH_PHY_ADDRESS

IS_ETH_AUTONEGOTIATION

IS_ETH_SPEED

IS_ETH_DUPLEX_MODE

IS_ETH_DUPLEX_MODE

IS_ETH_RX_MODE

IS_ETH_RX_MODE

IS_ETH_RX_MODE

IS_ETH_CHECKSUM_MODE

IS_ETH_MEDIA_INTERFACE

IS_ETH_WATCHDOG

IS_ETH_JABBER

IS_ETH_INTER_FRAME_GAP

IS_ETH_CARRIER_SENSE

IS_ETH_RECEIVE_OWN

IS_ETH_LOOPBACK_MODE

IS_ETH_CHECKSUM_OFFLOAD

IS_ETH_RETRY_TRANSMISSION

IS_ETH_AUTOMATIC_PADCRC_STRIP

IS_ETH_BACKOFF_LIMIT

IS_ETH_DEFERRAL_CHECK

IS_ETH_RECEIVE_ALL

IS_ETH_SOURCE_ADDR_FILTER

IS_ETH_CONTROL_FRAMES

IS_ETH_BROADCAST_FRAMES_RECEPTION

IS_ETH_DESTINATION_ADDR_FILTER

IS_ETH_PROMISCUOUS_MODE

IS_ETH_MULTICAST_FRAMES_FILTER

IS_ETH_UNICAST_FRAMES_FILTER

IS_ETH_PAUSE_TIME

IS_ETH_ZEROQUANTA_PAUSE

IS_ETH_PAUSE_LOW_THRESHOLD

IS_ETH_UNICAST_PAUSE_FRAME_DETECT

IS_ETH_RECEIVE_FLOWCONTROL

IS_ETH_TRANSMIT_FLOWCONTROL

IS_ETH_VLAN_TAG_COMPARISON

IS_ETH_VLAN_TAG_IDENTIFIER

IS_ETH_MAC_ADDRESS0123

IS_ETH_MAC_ADDRESS123

IS_ETH_MAC_ADDRESS_FILTER

IS_ETH_MAC_ADDRESS_MASK

IS_ETH_DROP_TCPIP_CHECKSUM_FRAME

IS_ETH_RECEIVE_STORE_FORWARD

IS_ETH_FLUSH_RECEIVE_FRAME

IS_ETH_TRANSMIT_STORE_FORWARD

IS_ETH_TRANSMIT_THRESHOLD_CONTROL

IS_ETH_FORWARD_ERROR_FRAMES

IS_ETH_FORWARD_UNDERSIZED_GOOD_FRAMES

IS_ETH_RECEIVE_THRESHOLD_CONTROL

IS_ETH_SECOND_FRAME_OPERATE

IS_ETH_ADDRESS_ALIGNED_BEATS

IS_ETH_FIXED_BURST

IS_ETH_RXDMA_BURST_LENGTH

IS_ETH_TXDMA_BURST_LENGTH

IS_ETH_DMA_DESC_SKIP_LENGTH

IS_ETH_DMA_ARBITRATION_ROUNDROBIN_RXTX

IS_ETH_DMA_TXDESC_SEGMENT

IS_ETH_DMA_TXDESC_CHECKSUM

IS_ETH_DMATXDESC_BUFFER_SIZE

IS_ETH_DMA_RXDESC_BUFFER

IS_ETH_DMA_GET_OVERFLOW

ETH Promiscuous Mode

ETH_PROMISCUOUS_MODE_ENABLE

ETH_PROMISCUOUS_MODE_DISABLE

ETH Receive All

ETH_RECEIVEALL_ENABLE

ETH_RECEIVEAll_DISABLE

ETH Receive Flow Control

ETH_RECEIVEFLOWCONTROL_ENABLE

ETH_RECEIVEFLOWCONTROL_DISABLE

ETH Receive Own

ETH_RECEIVEOWN_ENABLE

ETH_RECEIVEOWN_DISABLE

ETH Receive Store Forward

ETH_RECEIVESTOREFORWARD_ENABLE

ETH_RECEIVESTOREFORWARD_DISABLE

ETH Receive Threshold Control

ETH_RECEIVEDTHRESHOLDCONTROL_64BYTES	threshold level of the MTL Receive FIFO is 64 Bytes
ETH_RECEIVEDTHRESHOLDCONTROL_32BYTES	threshold level of the MTL Receive FIFO is 32 Bytes
ETH_RECEIVEDTHRESHOLDCONTROL_96BYTES	threshold level of the MTL Receive FIFO is 96 Bytes
ETH_RECEIVEDTHRESHOLDCONTROL_128BYTES	threshold level of the MTL Receive FIFO is 128 Bytes

ETH Retry Transmission

ETH_RETRYTRANSMISSION_ENABLE

ETH_RETRYTRANSMISSION_DISABLE

ETH Rx DMA_Burst Length

ETH_RXDMABURSTLENGTH_1BEAT	maximum number of beats to be transferred in one RxDMA transaction is 1
ETH_RXDMABURSTLENGTH_2BEAT	maximum number of beats to be transferred in one RxDMA transaction is 2
ETH_RXDMABURSTLENGTH_4BEAT	maximum number of beats to be transferred in one RxDMA transaction is 4
ETH_RXDMABURSTLENGTH_8BEAT	maximum number of beats to be transferred in one RxDMA transaction is 8
ETH_RXDMABURSTLENGTH_16BEAT	maximum number of beats to be transferred in one RxDMA transaction is 16
ETH_RXDMABURSTLENGTH_32BEAT	maximum number of beats to be transferred in one RxDMA transaction is 32
ETH_RXDMABURSTLENGTH_4XPBL_4BEAT	maximum number of beats to be transferred in one RxDMA transaction is 4
ETH_RXDMABURSTLENGTH_4XPBL_8BEAT	maximum number of beats to be transferred in one RxDMA transaction is 8
ETH_RXDMABURSTLENGTH_4XPBL_16BEAT	maximum number of beats to be transferred in one RxDMA transaction

	is 16
ETH_RXDMABURSTLENGTH_4XPBL_32BEAT	maximum number of beats to be transferred in one RxDMA transaction is 32
ETH_RXDMABURSTLENGTH_4XPBL_64BEAT	maximum number of beats to be transferred in one RxDMA transaction is 64
ETH_RXDMABURSTLENGTH_4XPBL_128BEAT	maximum number of beats to be transferred in one RxDMA transaction is 128

ETH Rx Mode

ETH_RXPOLLING_MODE

ETH_RXINTERRUPT_MODE

ETH Second Frame Operate

ETH_SECONDFRAMEOPERARTE_ENABLE

ETH_SECONDFRAMEOPERARTE_DISABLE

ETH Source Addr Filter

ETH_SOURCEADDRFILTER_NORMAL_ENABLE

ETH_SOURCEADDRFILTER_INVERSE_ENABLE

ETH_SOURCEADDRFILTER_DISABLE

ETH Speed

ETH_SPEED_10M

ETH_SPEED_100M

ETH Transmit Flow Control

ETH_TRANSMITFLOWCONTROL_ENABLE

ETH_TRANSMITFLOWCONTROL_DISABLE

ETH Transmit Store Forward

ETH_TRANSMITSTOREFORWARD_ENABLE

ETH_TRANSMITSTOREFORWARD_DISABLE

ETH Transmit Threshold Control

ETH_TRANSMITTHRESHOLDCONTROL_64BYTES	threshold level of the MTL Transmit FIFO is 64 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_128BYTES	threshold level of the MTL Transmit FIFO is 128 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_192BYTES	threshold level of the MTL Transmit FIFO is 192 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_256BYTES	threshold level of the MTL Transmit FIFO is 256 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_40BYTES	threshold level of the MTL Transmit FIFO is 40 Bytes

ETH_TRANSMITTHRESHOLDCONTROL_32BYTES	threshold level of the MTL Transmit FIFO is 32 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_24BYTES	threshold level of the MTL Transmit FIFO is 24 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_16BYTES	threshold level of the MTL Transmit FIFO is 16 Bytes

ETH Tx DMA Burst Length

ETH_TXDMABURSTLENGTH_1BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 1
ETH_TXDMABURSTLENGTH_2BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 2
ETH_TXDMABURSTLENGTH_4BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 4
ETH_TXDMABURSTLENGTH_8BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 8
ETH_TXDMABURSTLENGTH_16BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 16
ETH_TXDMABURSTLENGTH_32BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 32
ETH_TXDMABURSTLENGTH_4XPBL_4BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 4
ETH_TXDMABURSTLENGTH_4XPBL_8BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 8
ETH_TXDMABURSTLENGTH_4XPBL_16BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 16
ETH_TXDMABURSTLENGTH_4XPBL_32BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 32
ETH_TXDMABURSTLENGTH_4XPBL_64BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 64
ETH_TXDMABURSTLENGTH_4XPBL_128BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 128

ETH Unicast Frames Filter

ETH_UNICASTFRAMESFILTER_PERFECTHASHTABLE

ETH_UNICASTFRAMESFILTER_HASHTABLE

ETH_UNICASTFRAMESFILTER_PERFECT

ETH Unicast Pause Frame Detect

ETH_UNICASTPAUSEFRAMEDETECT_ENABLE

ETH_UNICASTPAUSEFRAMEDETECT_DISABLE

ETH VLAN Tag Comparison

ETH_VLANTAGCOMPARISON_12BIT

ETH_VLANTAGCOMPARISON_16BIT

ETH Watchdog

ETH_WATCHDOG_ENABLE

ETH_WATCHDOG_DISABLE

ETH Zero Quanta Pause

ETH_ZEROQUANTAPAUSE_ENABLE

ETH_ZEROQUANTAPAUSE_DISABLE

16 HAL FLASH Generic Driver

16.1 FLASH Firmware driver registers structures

16.1.1 FLASH_ProcessTypeDef

FLASH_ProcessTypeDef is defined in the stm32f1xx_hal_flash.h

Data Fields

- ***__IO FLASH_ProcedureTypeDef ProcedureOnGoing***
- ***__IO uint32_t DataRemaining***
- ***__IO uint32_t Address***
- ***__IO uint64_t Data***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing***
- ***__IO uint32_t FLASH_ProcessTypeDef::DataRemaining***
- ***__IO uint32_t FLASH_ProcessTypeDef::Address***
- ***__IO uint64_t FLASH_ProcessTypeDef::Data***
- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
- ***__IO uint32_t FLASH_ProcessTypeDef::ErrorCode***

16.2 FLASH Firmware driver API description

The following section lists the various functions of the FLASH library.

16.2.1 FLASH peripheral features

The Flash memory interface manages CPU AHB I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The Flash memory interface accelerates code execution with a system of instruction prefetch.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Prefetch on I-Code
- Option Bytes programming

16.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32F1xx devices. These functions are split in 3 groups:

1. FLASH Memory I/O Programming functions: this group includes all needed functions to erase and program the main memory:
   - Lock and Unlock the FLASH interface
   - Erase function: Erase page, erase all pages
   - Program functions: half word, word and doubleword
2. Option Bytes Programming functions: this group includes all needed functions to manage the Option Bytes:
   - Lock and Unlock the Option Bytes
   - Erase Option Bytes
   - Set/Reset the write protection
   - Set the Read protection Level
   - Program the user Option Bytes
   - Program the data Option Bytes
   - Launch the Option Bytes loader
3. Interrupts and flags management functions : this group includes all needed functions to:
   - Handle FLASH interrupts
   - Wait for last FLASH operation according to its status
   - Get error flag status

In addition to these function, this driver includes a set of macros allowing to handle the following operations:

- Set the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the half cycle access
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

16.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations (write/erase).

- ***HAL_FLASH_Program()***
- ***HAL_FLASH_Program_IT()***
- ***HAL_FLASH_IRQHandler()***
- ***HAL_FLASH_EndOfOperationCallback()***
- ***HAL_FLASH_OperationErrorCallback()***

16.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

- ***HAL_FLASH_Unlock()***
- ***HAL_FLASH_Lock()***
- ***HAL_FLASH_OB_Unlock()***
- ***HAL_FLASH_OB_Lock()***
- ***HAL_FLASH_OB_Launch()***

16.2.5 Peripheral State functions

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

- *HAL_FLASH_GetError()*

16.2.6 HAL_FLASH_Program

Function Name	**HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t Data)**
Function Description	Program halfword, word or double word at a specified address.
Parameters	• **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of Type Program • **Address:** Specifies the address to be programmed. • **Data:** Specifies the data to be programmed
Return values	• HAL_StatusTypeDef HAL Status
Notes	• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface • If an erase and a program operations are requested simultaneously, the erase operation is performed before the program one. • FLASH should be previously erased before new programmation (only exception to this is when 0x0000 is programmed)

16.2.7 HAL_FLASH_Program_IT

Function Name	**HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)**
Function Description	Program halfword, word or double word 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 Type Program • **Address:** Specifies the address to be programmed. • **Data:** Specifies the data to be programmed
Return values	• HAL_StatusTypeDef HAL Status
Notes	• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface • If an erase and a program operations are requested simultaneously, the erase operation is performed before the program one.

16.2.8 HAL_FLASH_IRQHandler

Function Name	**void HAL_FLASH_IRQHandler (void)**
Function Description	This function handles FLASH interrupt request.
Return values	• None

16.2.9 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: No return value expectedPages Erase: Address of the page which has been erasedProgram: Address which was selected for data program
Return values	• none

16.2.10 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: No return value expectedPages Erase: Address of the page which returned an errorProgram: Address which was selected for data program
Return values	• none

16.2.11 HAL_FLASH_Unlock

Function Name	**HAL_StatusTypeDef HAL_FLASH_Unlock (void)**
Function Description	Unlock the FLASH control register access.
Return values	• HAL Status

16.2.12 HAL_FLASH_Lock

Function Name	**HAL_StatusTypeDef HAL_FLASH_Lock (void)**
Function Description	Locks the FLASH control register access.
Return values	• HAL Status

16.2.13 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

16.2.14 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

16.2.15 HAL_FLASH_OB_Launch

Function Name	**HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)**
Function Description	Launch the option byte loading.
Return values	• HAL_StatusTypeDef HAL Status
Notes	• This function will reset automatically the MCU.

16.2.16 HAL_FLASH_GetError

Function Name	**uint32_t HAL_FLASH_GetError (void)**
Function Description	Get the specific FLASH error flag.
Return values	• FLASH_ErrorCode The returned value can be: FLASH_ERROR_PG: FLASH Programming error flag FLASH_ERROR_WRP: FLASH Write protected error flag

16.3 FLASH Firmware driver defines

The following section lists the various define and macros of the module.

16.3.1 FLASH

FLASH

FLASH Error Codes

FLASH_ERROR_NONE

FLASH_ERROR_PG

FLASH_ERROR_WRP

FLASH_ERROR_OPTV

FLASH Exported Macros

__HAL_FLASH_HALF_CYCLE_ACCESS_ENABLE	**Description:** • Enable the FLASH half cycle access. **Return value:** • None:
__HAL_FLASH_HALF_CYCLE_ACCESS_DISABLE	**Description:** • Disable the FLASH half cycle access. **Return value:** • None:

Flag definition

FLASH_FLAG_BSY	FLASH Bank1 Busy flag
FLASH_FLAG_PGERR	FLASH Bank1 Programming error flag
FLASH_FLAG_WRPERR	FLASH Bank1 Write protected error flag
FLASH_FLAG_EOP	FLASH Bank1 End of Operation flag
FLASH_FLAG_BSY_BANK1	FLASH Bank1 Busy flag
FLASH_FLAG_PGERR_BANK1	FLASH Bank1 Programming error flag
FLASH_FLAG_WRPERR_BANK1	FLASH Bank1 Write protected error flag
FLASH_FLAG_EOP_BANK1	FLASH Bank1 End of Operation flag
FLASH_FLAG_BSY_BANK2	FLASH Bank2 Busy flag
FLASH_FLAG_PGERR_BANK2	FLASH Bank2 Programming error flag
FLASH_FLAG_WRPERR_BANK2	FLASH Bank2 Write protected error flag
FLASH_FLAG_EOP_BANK2	FLASH Bank2 End of Operation flag
FLASH_FLAG_OPTVERR	Option Byte Error

Interrupt

__HAL_FLASH_ENABLE_IT	**Description:** • Enable the specified FLASH interrupt. **Parameters:** • __INTERRUPT__: : FLASH interrupt This parameter can be any combination of the following values: – FLASH_IT_EOP_BANK1: End of FLASH Operation Interrupt on bank1 – FLASH_IT_ERR_BANK1: Error Interrupt on bank1 – FLASH_IT_EOP_BANK2: End of FLASH Operation Interrupt on bank2 – FLASH_IT_ERR_BANK2: Error Interrupt on bank2 **Return value:** • none:

__HAL_FLASH_DISABLE_IT

Description:

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: : FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP_BANK1: End of FLASH Operation Interrupt on bank1
  - FLASH_IT_ERR_BANK1: Error Interrupt on bank1
  - FLASH_IT_EOP_BANK2: End of FLASH Operation Interrupt on bank2
  - FLASH_IT_ERR_BANK2: Error Interrupt on bank2

Return value:

- none:

__HAL_FLASH_GET_FLAG

Description:

- Get the specified FLASH flag status.

Parameters:

- __FLAG__: specifies the FLASH flag to check. This parameter can be one of the following values:
  - FLASH_FLAG_EOP_BANK1 : FLASH End of Operation flag on bank1
  - FLASH_FLAG_WRPERR_BANK1: FLASH Write protected error flag on bank1
  - FLASH_FLAG_PGERR_BANK1 : FLASH Programming error flag on bank1
  - FLASH_FLAG_BSY_BANK1 : FLASH Busy flag on bank1
  - FLASH_FLAG_EOP_BANK2 : FLASH End of Operation flag on bank2
  - FLASH_FLAG_WRPERR_BANK2: FLASH Write protected error flag on bank2
  - FLASH_FLAG_PGERR_BANK2 : FLASH Programming error flag on bank2
  - FLASH_FLAG_BSY_BANK2 : FLASH Busy flag on bank2
  - FLASH_FLAG_OPTVERR : Loaded OB and its complement do not match

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_FLASH_CLEAR_FLAG

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 : FLASH End of Operation flag on bank1
- FLASH_FLAG_WRPERR_BANK1: FLASH Write protected error flag on bank1
- FLASH_FLAG_PGERR_BANK1 : FLASH Programming error flag on bank1
- FLASH_FLAG_BSY_BANK1 : FLASH Busy flag on bank1
- FLASH_FLAG_EOP_BANK2 : FLASH End of Operation flag on bank2
- FLASH_FLAG_WRPERR_BANK2: FLASH Write protected error flag on bank2
- FLASH_FLAG_PGERR_BANK2 : FLASH Programming error flag on bank2
- FLASH_FLAG_BSY_BANK2 : FLASH Busy flag on bank2
- FLASH_FLAG_OPTVERR : Loaded OB and its complement do not match

Return value:

- none:

Interrupt definition

FLASH_IT_EOP	End of FLASH Operation Interrupt source Bank1
FLASH_IT_ERR	Error Interrupt source Bank1
FLASH_IT_EOP_BANK1	End of FLASH Operation Interrupt source Bank1
FLASH_IT_ERR_BANK1	Error Interrupt source Bank1
FLASH_IT_EOP_BANK2	End of FLASH Operation Interrupt source Bank2
FLASH_IT_ERR_BANK2	Error Interrupt source Bank2

Latency configuration

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- __LATENCY__: FLASH Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0: FLASH Zero Latency cycle
  - FLASH_LATENCY_1: FLASH One Latency cycle
  - FLASH_LATENCY_2: FLASH Two Latency cycle

Return value:

- None:

__HAL_FLASH_GET_LATENCY

Description:

- Get the FLASH Latency.

Return value:

- FLASH: Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0: FLASH Zero Latency cycle
  - FLASH_LATENCY_1: FLASH One Latency cycle
  - FLASH_LATENCY_2: FLASH Two Latency cycle

Latency Values

FLASH_LATENCY_0	FLASH Zero Latency cycle
FLASH_LATENCY_1	FLASH One Latency cycle
FLASH_LATENCY_2	FLASH Two Latency cycles

Prefetch activation or deactivation

__HAL_FLASH_PREFETCH_BUFFER_ENABLE

Description:

- Enable the FLASH prefetch buffer.

Return value:

- None:

__HAL_FLASH_PREFETCH_BUFFER_DISABLE

Description:

- Disable the FLASH prefetch buffer.

Return value:

- None:

FLASH Private Constants

FLASH_TIMEOUT_VALUE

FLASH Private Macros

IS_FLASH_TYPEPROGRAM

Type Program

FLASH_TYPEPROGRAM_HALFWORD	Program a half-word (16-bit) at a specified address.
FLASH_TYPEPROGRAM_WORD	Program a word (32-bit) at a specified address.
FLASH_TYPEPROGRAM_DOUBLEWORD	Program a double word (64-bit) at a specified address

17 HAL FLASH Extension Driver

17.1 FLASHEx Firmware driver registers structures

17.1.1 FLASH_EraseInitTypeDef

FLASH_EraseInitTypeDef is defined in the stm32f1xx_hal_flash_ex.h

Data Fields

- ***uint32_t TypeErase***
- ***uint32_t Banks***
- ***uint32_t PageAddress***
- ***uint32_t NbPages***

Field Documentation

- ***uint32_t FLASH_EraseInitTypeDef::TypeErase*** TypeErase: Mass erase or page erase. This parameter can be a value of ***FLASHEx_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::Banks*** Select banks to erase when Mass erase is enabled. This parameter must be a value of ***FLASHEx_Banks***
- ***uint32_t FLASH_EraseInitTypeDef::PageAddress*** PageAdress: Initial FLASH page address to erase when mass erase is disabled This parameter must be a number between Min_Data = 0x08000000 and Max_Data = FLASH_BANKx_END (x = 1 or 2 depending on devices)
- ***uint32_t FLASH_EraseInitTypeDef::NbPages*** NbPages: Number of pagess to be erased. This parameter must be a value between Min_Data = 1 and Max_Data = (max number of pages - value of initial page)

17.1.2 FLASH_OBProgramInitTypeDef

FLASH_OBProgramInitTypeDef is defined in the stm32f1xx_hal_flash_ex.h

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPState***
- ***uint32_t WRPPage***
- ***uint32_t Banks***
- ***uint8_t RDPLevel***
- ***uint8_t USERConfig***
- ***uint32_t DATAAddress***
- ***uint8_t DATAData***

Field Documentation

- ***uint32_t FLASH_OBProgramInitTypeDef::OptionType*** OptionType: Option byte to be configured. This parameter can be a value of ***FLASHEx_OB_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPState*** WRPState: Write protection activation or deactivation. This parameter can be a value of ***FLASHEx_OB_WRP_State***

- ***uint32_t FLASH_OBProgramInitTypeDef::WRPPage*** WRPPage: specifies the page(s) to be write protected This parameter can be a value of ***FLASHEx_OB_Write_Protection***
- ***uint32_t FLASH_OBProgramInitTypeDef::Banks*** Select banks for WRP activation/deactivation of all sectors. This parameter must be a value of ***FLASHEx_Banks***
- ***uint8_t FLASH_OBProgramInitTypeDef::RDPLevel*** RDPLevel: Set the read protection level.. This parameter can be a value of ***FLASHEx_OB_Read_Protection***
- ***uint8_t FLASH_OBProgramInitTypeDef::USERConfig*** USERConfig: Program the FLASH User Option Byte: IWDG / STOP / STDBY / BOOT1 This parameter can be a combination of ***FLASHEx_OB_IWatchdog***, ***FLASHEx_OB_nRST_STOP***, ***FLASHEx_OB_nRST_STDBY***, ***FLASHEx_OB_BOOT1***
- ***uint32_t FLASH_OBProgramInitTypeDef::DATAAddress*** DATAAddress: Address of the option byte DATA to be prgrammed This parameter can be a value of ***FLASHEx_OB_Data_Address***
- ***uint8_t FLASH_OBProgramInitTypeDef::DATAData*** DATAData: Data to be stored in the option byte DATA This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF

17.2 FLASHEx Firmware driver API description

The following section lists the various functions of the FLASHEx library.

17.2.1 IO operation functions

- ***HAL_FLASHEx_Erase()***
- ***HAL_FLASHEx_Erase_IT()***

17.2.2 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

- ***HAL_FLASHEx_OBErase()***
- ***HAL_FLASHEx_OBProgram()***
- ***HAL_FLASHEx_OBGetConfig()***

17.2.3 HAL_FLASHEx_Erase

Function Name	**HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)**
Function Description	Perform a mass erase or erase the specified FLASH memory pages.
Parameters	• **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing. • **PageError:** pointer to variable that contains the configuration information on faulty page in case of error (0xFFFFFFFF means that all the pages have been correctly erased)

Return values	• HAL_StatusTypeDef HAL Status
Notes	• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface

17.2.4 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_StatusTypeDef HAL Status
Notes	• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_Lock() should be called after to lock the FLASH interface

17.2.5 HAL_FLASHEx_OBErase

Function Name	**HAL_StatusTypeDef HAL_FLASHEx_OBErase (void)**
Function Description	Erases the FLASH option bytes.
Return values	• HAL status
Notes	• This functions erases all option bytes except the Read protection (RDP). The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_OB_Unlock() should be called before to unlock the options bytes The function HAL_FLASH_OB_Launch() should be called after to force the reload of the options bytes (system reset will occur)

17.2.6 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_StatusTypeDef HAL Status
Notes	• The function HAL_FLASH_Unlock() should be called before to unlock the FLASH interface The function HAL_FLASH_OB_Unlock() should be called before to unlock

	the options bytes The function HAL_FLASH_OB_Launch() should be called after to force the reload of the options bytes (system reset will occur)

17.2.7 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

17.3 FLASHEx Firmware driver defines

The following section lists the various define and macros of the module.

17.3.1 FLASHEx

FLASHEx

Banks

FLASH_BANK_1	Bank 1
FLASH_BANK_2	Bank 2
FLASH_BANK_BOTH	Bank1 and Bank2

Option Byte BOOT1

OB_BOOT1_RESET	BOOT1 Reset
OB_BOOT1_SET	BOOT1 Set

Option Byte Data Address

OB_DATA_ADDRESS_DATA0

OB_DATA_ADDRESS_DATA1

Option Byte IWatchdog

OB_IWDG_SW	Software IWDG selected
OB_IWDG_HW	Hardware IWDG selected

Option Byte nRST STDBY

OB_STDBY_NO_RST	No reset generated when entering in STANDBY
OB_STDBY_RST	Reset generated when entering in STANDBY

Option Byte nRST STOP

OB_STOP_NO_RST	No reset generated when entering in STOP
OB_STOP_RST	Reset generated when entering in STOP

Option Byte Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_1

Option Bytes Type

OPTIONBYTE_WRP	WRP option byte configuration
OPTIONBYTE_RDP	RDP option byte configuration
OPTIONBYTE_USER	USER option byte configuration
OPTIONBYTE_DATA	DATA option byte configuration

Option Bytes Write Protection

OB_WRP_PAGES0TO1	Write protection of page 0 TO 1
OB_WRP_PAGES2TO3	Write protection of page 2 TO 3
OB_WRP_PAGES4TO5	Write protection of page 4 TO 5
OB_WRP_PAGES6TO7	Write protection of page 6 TO 7
OB_WRP_PAGES8TO9	Write protection of page 8 TO 9
OB_WRP_PAGES10TO11	Write protection of page 10 TO 11
OB_WRP_PAGES12TO13	Write protection of page 12 TO 13
OB_WRP_PAGES14TO15	Write protection of page 14 TO 15
OB_WRP_PAGES16TO17	Write protection of page 16 TO 17
OB_WRP_PAGES18TO19	Write protection of page 18 TO 19
OB_WRP_PAGES20TO21	Write protection of page 20 TO 21
OB_WRP_PAGES22TO23	Write protection of page 22 TO 23
OB_WRP_PAGES24TO25	Write protection of page 24 TO 25
OB_WRP_PAGES26TO27	Write protection of page 26 TO 27
OB_WRP_PAGES28TO29	Write protection of page 28 TO 29
OB_WRP_PAGES30TO31	Write protection of page 30 TO 31
OB_WRP_PAGES32TO33	Write protection of page 32 TO 33
OB_WRP_PAGES34TO35	Write protection of page 34 TO 35
OB_WRP_PAGES36TO37	Write protection of page 36 TO 37
OB_WRP_PAGES38TO39	Write protection of page 38 TO 39
OB_WRP_PAGES40TO41	Write protection of page 40 TO 41
OB_WRP_PAGES42TO43	Write protection of page 42 TO 43
OB_WRP_PAGES44TO45	Write protection of page 44 TO 45
OB_WRP_PAGES46TO47	Write protection of page 46 TO 47
OB_WRP_PAGES48TO49	Write protection of page 48 TO 49
OB_WRP_PAGES50TO51	Write protection of page 50 TO 51
OB_WRP_PAGES52TO53	Write protection of page 52 TO 53
OB_WRP_PAGES54TO55	Write protection of page 54 TO 55
OB_WRP_PAGES56TO57	Write protection of page 56 TO 57

OB_WRP_PAGES58TO59	Write protection of page 58 TO 59
OB_WRP_PAGES60TO61	Write protection of page 60 TO 61
OB_WRP_PAGES62TO127	Write protection of page 62 TO 127
OB_WRP_PAGES62TO255	Write protection of page 62 TO 255
OB_WRP_PAGES62TO511	Write protection of page 62 TO 511
OB_WRP_ALLPAGES	Write protection of all Pages
OB_WRP_PAGES0TO15MASK	
OB_WRP_PAGES16TO31MASK	
OB_WRP_PAGES32TO47MASK	
OB_WRP_PAGES48TO511MASK	

Option Byte WRP State

OB_WRPSTATE_DISABLE	Disable the write protection of the desired pages
OB_WRPSTATE_ENABLE	Enable the write protection of the desired pagess

FLASHEx Private Constants

FLASH_SIZE_DATA_REGISTER

OBR_REG_INDEX

SR_FLAG_MASK

FLASH_PAGE_SIZE

FLASHEx Private Macros

IS_FLASH_TYPEERASE

IS_OPTIONBYTE

IS_WRPSTATE

IS_OB_RDP_LEVEL

IS_OB_DATA_ADDRESS

IS_OB_IWDG_SOURCE

IS_OB_STOP_SOURCE

IS_OB_STDBY_SOURCE

IS_OB_BOOT1

IS_FLASH_NB_PAGES

IS_OB_WRP

IS_FLASH_BANK

IS_FLASH_PROGRAM_ADDRESS

IS_FLASH_LATENCY

Type Erase

FLASH_TYPEERASE_PAGES	Pages erase only
FLASH_TYPEERASE_MASSERASE	Flash mass erase activation

18 HAL GPIO Generic Driver

18.1 GPIO Firmware driver registers structures

18.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32f1xx_hal_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Pull***
- ***uint32_t Speed***

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***

18.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

18.2.1 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 20 edge detectors in connectivity line devices, or 19 edge detectors in other devices for generating event/interrupt requests. Each input line can be independently configured to select the type (event or interrupt) and the corresponding trigger event (rising or falling or both). Each line can also masked independently. A pending register maintains the status line of the interrupt requests

18.2.2 How to use this driver

1. Enable the GPIO APB2 clock using the following function : __HAL_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
   - 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 PD0 and PD1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

18.2.3 Initialization and deinitialization functions

This section provides functions allowing to initialize and de-initialize the GPIOs to be ready for use.

- ***HAL_GPIO_Init()***

- *HAL_GPIO_DeInit()*

18.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the GPIOs.

- *HAL_GPIO_ReadPin()*
- *HAL_GPIO_WritePin()*
- *HAL_GPIO_TogglePin()*
- *HAL_GPIO_LockPin()*
- *HAL_GPIO_EXTI_IRQHandler()*
- *HAL_GPIO_EXTI_Callback()*

18.2.5 HAL_GPIO_Init

Function Name	**void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)**
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• None

18.2.6 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..G depending on device used) 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

18.2.7 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..G depending on device used) 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.

18.2.8 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..G depending on device used) 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_BIT_RESET: to clear the port pin GPIO_BIT_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.

18.2.9 HAL_GPIO_TogglePin

Function Name	**void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function Description	Toggles the specified GPIO pin.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Pin:** Specifies the pins to be toggled.
Return values	• None

18.2.10 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..G depending on device used) to select the GPIO peripheral • **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 locking mechanism allows the IO configuration to be frozen. When the LOCK sequence has been applied on a port bit, it is no longer possible to modify the value of the port bit until the next reset.

18.2.11 HAL_GPIO_EXTI_IRQHandler

Function Name	**void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)**
Function Description	This function handles EXTI interrupt request.
Parameters	• **GPIO_Pin:** Specifies the pins connected EXTI line
Return values	• None

18.2.12 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 pins connected EXTI line
Return values	• None

18.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

18.3.1 GPIO

GPIO

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_AF_INPUT	Alternate Function Input 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 Private Constants

GPIO_MODE

EXTI_MODE

GPIO_MODE_IT

GPIO_MODE_EVT

RISING_EDGE

FALLING_EDGE

GPIO_OUTPUT_TYPE	
GPIO_NUMBER	
GPIO_CR_MODE_INPUT	00: Input mode (reset state)
GPIO_CR_CNF_ANALOG	00: Analog mode
GPIO_CR_CNF_INPUT_FLOATING	01: Floating input (reset state)
GPIO_CR_CNF_INPUT_PU_PD	10: Input with pull-up / pull-down
GPIO_CR_CNF_GP_OUTPUT_PP	00: General purpose output push-pull
GPIO_CR_CNF_GP_OUTPUT_OD	01: General purpose output Open-drain
GPIO_CR_CNF_AF_OUTPUT_PP	10: Alternate function output Push-pull
GPIO_CR_CNF_AF_OUTPUT_OD	11: Alternate function output Open-drain

GPIO_Private_Macros

IS_GPIO_PIN_ACTION

IS_GPIO_PIN

IS_GPIO_PULL

IS_GPIO_SPEED

IS_GPIO_MODE

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_LOW	Low speed
GPIO_SPEED_MEDIUM	Medium speed
GPIO_SPEED_HIGH	High speed

19 HAL GPIO Extension Driver

19.1 GPIOEx Firmware driver API description

The following section lists the various functions of the GPIOEx library.

19.1.1 GPIO Peripheral extension features

GPIO module on STM32F1 family, manage also the AFIO register:

- Possibility to use the EVENTOUT Cortex feature

19.1.2 How to use this driver

This driver provides functions to use EVENTOUT Cortex feature

1. Configure EVENTOUT Cortex feature using the function HAL_GPIOEx_ConfigEventout()
2. Activate EVENTOUT Cortex feature using the HAL_GPIOEx_EnableEventout()
3. Deactivate EVENTOUT Cortex feature using the HAL_GPIOEx_DisableEventout()

19.1.3 Extended features functions

This section provides functions allowing to:

- Configure EVENTOUT Cortex feature using the function HAL_GPIOEx_ConfigEventout()
- Activate EVENTOUT Cortex feature using the HAL_GPIOEx_EnableEventout()
- Deactivate EVENTOUT Cortex feature using the HAL_GPIOEx_DisableEventout()
- ***HAL_GPIOEx_ConfigEventout()***
- ***HAL_GPIOEx_EnableEventout()***
- ***HAL_GPIOEx_DisableEventout()***

19.1.4 HAL_GPIOEx_ConfigEventout

Function Name	**void HAL_GPIOEx_ConfigEventout (uint32_t GPIO_PortSource, uint32_t GPIO_PinSource)**
Function Description	Configures the port and pin on which the EVENTOUT Cortex signal will be connected.
Parameters	• **GPIO_PortSource:** Select the port used to output the Cortex EVENTOUT signal. This parameter can be a value of EVENTOUT Port. • **GPIO_PinSource:** Select the pin used to output the Cortex EVENTOUT signal. This parameter can be a value of EVENTOUT Pin.
Return values	• None

19.1.5 HAL_GPIOEx_EnableEventout

Function Name	**void HAL_GPIOEx_EnableEventout (void)**
Function Description	Enables the Event Output.
Return values	• None

19.1.6 HAL_GPIOEx_DisableEventout

Function Name	**void HAL_GPIOEx_DisableEventout (void)**
Function Description	Disables the Event Output.
Return values	• None

19.2 GPIOEx Firmware driver defines

The following section lists the various define and macros of the module.

19.2.1 GPIOEx

GPIOEx

Alternate Function Remapping

__HAL_AFIO_REMAP_SPI1_ENABLE

Description:

- Enable the remapping of SPI1 alternate function NSS, SCK, MISO and MOSI.

Return value:

- None:

__HAL_AFIO_REMAP_SPI1_DISABLE

Description:

- Disable the remapping of SPI1 alternate function NSS, SCK, MISO and MOSI.

Return value:

- None:

__HAL_AFIO_REMAP_I2C1_ENABLE

Description:

- Enable the remapping of I2C1 alternate function SCL and SDA.

Return value:

- None:

__HAL_AFIO_REMAP_I2C1_DISABLE

Description:

- Disable the remapping of I2C1 alternate function SCL and SDA.

Return value:

- None:

__HAL_AFIO_REMAP_USART1_ENABLE

Description:

- Enable the remapping of USART1 alternate function TX and RX.

Return value:

- None:

__HAL_AFIO_REMAP_USART1_DISABLE

Description:

- Disable the remapping of USART1 alternate function TX and RX.

Return value:

- None:

__HAL_AFIO_REMAP_USART2_ENABLE

Description:

- Enable the remapping of USART2 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None:

__HAL_AFIO_REMAP_USART2_DISABLE

Description:

- Disable the remapping of USART2 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None:

__HAL_AFIO_REMAP_USART3_ENABLE

Description:

- Enable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None:

__HAL_AFIO_REMAP_USART3_PARTIAL

Description:

- Enable the remapping of USART3 alternate function CTS, RTS, CK, TX and RX.

Return value:

- None:

__HAL_AFIO_REMAP_USART3_DISABLE

Description:

- Disable the remapping of USART3 alternate function

CTS, RTS, CK, TX and RX.

Return value:

- None:

__HAL_AFIO_REMAP_TIM1_ENABLE

Description:

- Enable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)

Return value:

- None:

__HAL_AFIO_REMAP_TIM1_PARTIAL

Description:

- Enable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)

Return value:

- None:

__HAL_AFIO_REMAP_TIM1_DISABLE

Description:

- Disable the remapping of TIM1 alternate function channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN)

Return value:

- None:

__HAL_AFIO_REMAP_TIM2_ENABLE

Description:

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return value:

- None:

__HAL_AFIO_REMAP_TIM2_PARTIAL_2

Description:

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return value:

- None:

__HAL_AFIO_REMAP_TIM2_PARTIAL_1

Description:

- Enable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return value:

- None:

__HAL_AFIO_REMAP_TIM2_DISABLE

Description:

- Disable the remapping of TIM2 alternate function channels 1 to 4 and external trigger (ETR)

Return value:

- None:

__HAL_AFIO_REMAP_TIM3_ENABLE

Description:

- Enable the remapping of TIM3 alternate function channels 1 to 4.

Return value:

- None:

__HAL_AFIO_REMAP_TIM3_PARTIAL

Description:

- Enable the remapping of TIM3 alternate function channels 1 to 4.

Return value:

- None:

__HAL_AFIO_REMAP_TIM3_DISABLE

Description:

- Disable the remapping of TIM3 alternate function channels 1 to 4.

Return value:

- None:

__HAL_AFIO_REMAP_TIM4_ENABLE

Description:

- Enable the remapping of TIM4 alternate function channels 1 to 4.

Return value:

- None:

__HAL_AFIO_REMAP_TIM4_DISABLE

Description:

- Disable the remapping of TIM4 alternate function channels 1 to 4.

Return value:

- None:

__HAL_AFIO_REMAP_CAN1_1

Description:

- Enable or disable the

remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return value:

- None:

__HAL_AFIO_REMAP_CAN1_2

Description:

- Enable or disable the remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return value:

- None:

__HAL_AFIO_REMAP_CAN1_3

Description:

- Enable or disable the remapping of CAN alternate function CAN_RX and CAN_TX in devices with a single CAN interface.

Return value:

- None:

__HAL_AFIO_REMAP_PD01_ENABLE

Description:

- Enable the remapping of PD0 and PD1.

Return value:

- None:

__HAL_AFIO_REMAP_PD01_DISABLE

Description:

- Disable the remapping of PD0 and PD1.

Return value:

- None:

__HAL_AFIO_REMAP_TIM5CH4_ENABLE

Description:

- Enable the remapping of TIM5CH4.

Return value:

- None:

__HAL_AFIO_REMAP_TIM5CH4_DISABLE

Description:

- Disable the remapping of TIM5CH4.

Return value:

- None:

__HAL_AFIO_REMAP_ADC1_ETRGINJ_ENABLE

Description:

- Enable the remapping of ADC1_ETRGINJ (ADC 1 External trigger injected conversion).

Return value:

- None:

__HAL_AFIO_REMAP_ADC1_ETRGINJ_DISABLE

Description:

- Disable the remapping of ADC1_ETRGINJ (ADC 1 External trigger injected conversion).

Return value:

- None:

__HAL_AFIO_REMAP_ADC1_ETRGREG_ENABLE

Description:

- Enable the remapping of ADC1_ETRGREG (ADC 1 External trigger regular conversion).

Return value:

- None:

__HAL_AFIO_REMAP_ADC1_ETRGREG_DISABLE

Description:

- Disable the remapping of ADC1_ETRGREG (ADC 1 External trigger regular conversion).

Return value:

- None:

__HAL_AFIO_REMAP_ADC2_ETRGINJ_ENABLE

Description:

- Enable the remapping of ADC2_ETRGREG (ADC 2 External trigger injected conversion).

Return value:

- None:

__HAL_AFIO_REMAP_ADC2_ETRGINJ_DISABLE

Description:

- Disable the remapping of ADC2_ETRGREG (ADC 2 External trigger injected conversion).

Return value:

- None:

__HAL_AFIO_REMAP_ADC2_ETRGREG_ENABLE

Description:

- Enable the remapping of ADC2_ETRGREG (ADC 2 External trigger regular conversion).

Return value:

- None:

__HAL_AFIO_REMAP_ADC2_ETRGREG_DISABLE

Description:

- Disable the remapping of ADC2_ETRGREG (ADC 2 External trigger regular conversion).

Return value:

- None:

__HAL_AFIO_REMAP_SWJ_ENABLE

Description:

- Enable the Serial wire JTAG configuration.

Return value:

- None:

__HAL_AFIO_REMAP_SWJ_NONJTRST

Description:

- Enable the Serial wire JTAG configuration.

Return value:

- None:

__HAL_AFIO_REMAP_SWJ_NOJTAG

Description:

- Enable the Serial wire JTAG configuration.

Return value:

- None:

__HAL_AFIO_REMAP_SWJ_DISABLE

Description:

- Disable the Serial wire JTAG configuration.

Return value:

- None:

__HAL_AFIO_REMAP_TIM9_ENABLE

Description:

- Enable the remapping of TIM9_CH1 and TIM9_CH2.

Return value:

- None:

__HAL_AFIO_REMAP_TIM9_DISABLE	**Description:** • Disable the remapping of TIM9_CH1 and TIM9_CH2. **Return value:** • None:
__HAL_AFIO_REMAP_TIM10_ENABLE	**Description:** • Enable the remapping of TIM10_CH1. **Return value:** • None:
__HAL_AFIO_REMAP_TIM10_DISABLE	**Description:** • Disable the remapping of TIM10_CH1. **Return value:** • None:
__HAL_AFIO_REMAP_TIM11_ENABLE	**Description:** • Enable the remapping of TIM11_CH1. **Return value:** • None:
__HAL_AFIO_REMAP_TIM11_DISABLE	**Description:** • Disable the remapping of TIM11_CH1. **Return value:** • None:
__HAL_AFIO_REMAP_TIM13_ENABLE	**Description:** • Enable the remapping of TIM13_CH1. **Return value:** • None:
__HAL_AFIO_REMAP_TIM13_DISABLE	**Description:** • Disable the remapping of TIM13_CH1. **Return value:** • None:
__HAL_AFIO_REMAP_TIM14_ENABLE	**Description:** • Enable the remapping of TIM14_CH1.

	Return value: • None:
__HAL_AFIO_REMAP_TIM14_DISABLE	**Description:** • Disable the remapping of TIM14_CH1. **Return value:** • None:
__HAL_AFIO_FSMCNADV_DISCONNECTED	**Description:** • Controls the use of the optional FSMC_NADV signal. **Return value:** • None:
__HAL_AFIO_FSMCNADV_CONNECTED	**Description:** • Controls the use of the optional FSMC_NADV signal. **Return value:** • None:

EVENTOUT Pin

AFIO_EVENTOUT_PIN_0	EVENTOUT on pin 0
AFIO_EVENTOUT_PIN_1	EVENTOUT on pin 1
AFIO_EVENTOUT_PIN_2	EVENTOUT on pin 2
AFIO_EVENTOUT_PIN_3	EVENTOUT on pin 3
AFIO_EVENTOUT_PIN_4	EVENTOUT on pin 4
AFIO_EVENTOUT_PIN_5	EVENTOUT on pin 5
AFIO_EVENTOUT_PIN_6	EVENTOUT on pin 6
AFIO_EVENTOUT_PIN_7	EVENTOUT on pin 7
AFIO_EVENTOUT_PIN_8	EVENTOUT on pin 8
AFIO_EVENTOUT_PIN_9	EVENTOUT on pin 9
AFIO_EVENTOUT_PIN_10	EVENTOUT on pin 10
AFIO_EVENTOUT_PIN_11	EVENTOUT on pin 11
AFIO_EVENTOUT_PIN_12	EVENTOUT on pin 12
AFIO_EVENTOUT_PIN_13	EVENTOUT on pin 13
AFIO_EVENTOUT_PIN_14	EVENTOUT on pin 14
AFIO_EVENTOUT_PIN_15	EVENTOUT on pin 15
IS_AFIO_EVENTOUT_PIN	

EVENTOUT Port

AFIO_EVENTOUT_PORT_A	EVENTOUT on port A

AFIO_EVENTOUT_PORT_B	EVENTOUT on port B
AFIO_EVENTOUT_PORT_C	EVENTOUT on port C
AFIO_EVENTOUT_PORT_D	EVENTOUT on port D
AFIO_EVENTOUT_PORT_E	EVENTOUT on port E

IS_AFIO_EVENTOUT_PORT

GPIOEx Private Macros

GPIO_GET_INDEX

20 HAL HCD Generic Driver

20.1 HCD Firmware driver registers structures

20.1.1 HCD_HandleTypeDef

HCD_HandleTypeDef is defined in the stm32f1xx_hal_hcd.h

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

20.2 HCD Firmware driver API description

The following section lists the various functions of the HCD library.

20.2.1 How to use this driver

1. Declare a HCD_HandleTypeDef handle structure, for example: HCD_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_HCD_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL_HCD_MspInit() API:
   a. Enable the HCD/USB Low Level interface clock using the following macro
      – __HAL_RCC_OTGFS_CLK_ENABLE()
   b. Initialize the related GPIO clocks
   c. Configure HCD pin-out
   d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
   a. hhcd.pData = phost;
6. Enable HCD transmission and reception:
   a. HAL_HCD_Start();

20.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- *HAL_HCD_Init()*
- *HAL_HCD_HC_Init()*
- *HAL_HCD_HC_Halt()*
- *HAL_HCD_DeInit()*
- *HAL_HCD_MspInit()*
- *HAL_HCD_MspDeInit()*

20.2.3 IO operation functions

- *HAL_HCD_HC_SubmitRequest()*
- *HAL_HCD_IRQHandler()*
- *HAL_HCD_SOF_Callback()*
- *HAL_HCD_Connect_Callback()*
- *HAL_HCD_Disconnect_Callback()*
- *HAL_HCD_HC_NotifyURBChange_Callback()*

20.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.

- *HAL_HCD_Start()*
- *HAL_HCD_Stop()*
- *HAL_HCD_ResetPort()*

20.2.5 Peripheral State functions

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

- *HAL_HCD_GetState()*
- *HAL_HCD_HC_GetURBState()*
- *HAL_HCD_HC_GetXferCount()*
- *HAL_HCD_HC_GetState()*
- *HAL_HCD_GetCurrentFrame()*
- *HAL_HCD_GetCurrentSpeed()*

20.2.6 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

20.2.7 HAL_HCD_HC_Init

Function Name	**HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)**
Function Description	Initialize a host channel.
Parameters	• **hhcd:** HCD handle • **ch_num:** Channel number. This parameter can be a value from 1 to 15 • **epnum:** Endpoint number. This parameter can be a value from 1 to 15 • **dev_address:** : Current device address This parameter can be a value from 0 to 255 • **speed:** Current device speed. This parameter can be one of these values: HCD_SPEED_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

20.2.8 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

20.2.9 HAL_HCD_DeInit

Function Name	**HAL_StatusTypeDef HAL_HCD_DeInit (HCD_HandleTypeDef * hhcd)**
Function Description	DeInitialize the host driver.
Parameters	• **hhcd:** HCD handle
Return values	• HAL status

20.2.10 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

20.2.11 HAL_HCD_MspDeInit

Function Name	**void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)**
Function Description	DeInitializes HCD MSP.
Parameters	• **hhcd:** HCD handle
Return values	• None

20.2.12 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

20.2.13 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

20.2.14 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

20.2.15 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

20.2.16 HAL_HCD_Disconnect_Callback

Function Name	**void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)**
Function Description	Disonnexion Event callback.
Parameters	• **hhcd:** HCD handle
Return values	• None

20.2.17 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

20.2.18 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

20.2.19 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

20.2.20 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

20.2.21 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

20.2.22 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/

20.2.23 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

20.2.24 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/

20.2.25 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

20.2.26 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

20.3 HCD Firmware driver defines

The following section lists the various define and macros of the module.

20.3.1 HCD

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 Speed

HCD_SPEED_LOW

HCD_SPEED_FULL

21 HAL I2C Generic Driver

21.1 I2C Firmware driver registers structures

21.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32f1xx_hal_i2c.h

Data Fields

- ***uint32_t ClockSpeed***
- ***uint32_t DutyCycle***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::ClockSpeed*** Specifies the clock frequency. This parameter must be set to a value lower than 400kHz
- ***uint32_t I2C_InitTypeDef::DutyCycle*** Specifies the I2C fast mode duty cycle. This parameter can be a value of ***I2C_duty_cycle_in_fast_mode***
- ***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::GeneralCallMode*** Specifies if general call mode is selected. This parameter can be a value of ***I2C_general_call_addressing_mode***
- ***uint32_t I2C_InitTypeDef::NoStretchMode*** Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_nostretch_mode***

21.1.2 I2C_HandleTypeDef

I2C_HandleTypeDef is defined in the stm32f1xx_hal_i2c.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***

- ***__IO HAL_I2C_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***I2C_TypeDef* I2C_HandleTypeDef::Instance*** I2C registers base address
- ***I2C_InitTypeDef I2C_HandleTypeDef::Init*** I2C communication parameters
- ***uint8_t* I2C_HandleTypeDef::pBuffPtr*** Pointer to I2C transfer buffer
- ***uint16_t I2C_HandleTypeDef::XferSize*** I2C transfer size
- ***__IO uint16_t I2C_HandleTypeDef::XferCount*** I2C transfer counter
- ***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 uint32_t I2C_HandleTypeDef::ErrorCode***

21.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

21.2.1 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 implement the HAL_I2C_MspInit() API:
   a. Enable the I2Cx interface clock
   b. I2C pins configuration
      - Enable the clock for the I2C GPIOs
      - Configure I2C pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the I2Cx interrupt priority
      - Enable the NVIC I2C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
      - Enable the DMAx interface clock using
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx channel
      - Associate the initilalized DMA handle to the hi2c DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Configure the Communication Speed, Duty cycle, Addressing mode, Own Address1, Dual Addressing mode, Own Address2, 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 customed 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

- The I2C interrupts should have the highest priority in the application in order to make them uninterruptible.
- 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

Interrupt mode IO MEM operation

- The I2C interrupts should have the highest priority in the application in order to make them uninterruptible.
- Write an amount of data in no-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At MEM 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 no-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At MEM 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

DMA mode IO MEM operation

- Write an amount of data in no-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()
- At MEM 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 no-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At MEM 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

You can refer to the I2C HAL driver header file for more useful macros

I2C Workarounds linked to Silicon Limitation

See ErrataSheet to know full silicon limitation list of your product. (#) Workarounds Implemented inside I2C HAL Driver (##) Wrong data read into data register (Polling and Interrupt mode) (##) Start cannot be generated after a misplaced Stop (##) Some software events must be managed before the current byte is being transferred: Workaround: Use DMA in general, except when the Master is receiving a single byte. For Interupt mode, I2C should have the highest priority in the application. (##) Mismatch on the "Setup time for a repeated Start condition" timing parameter: Workaround: Reduce the frequency down to 88 kHz or use the I2C Fast-mode if supported by the slave. (##) Data valid time (tVD;DAT) violated without the OVR flag being set: Workaround: If the slave device allows it, use the clock stretching mechanism by programming NoStretchMode = I2C_NOSTRETCH_DISABLE in HAL_I2C_Init.

21.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze 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:
  - Communication Speed
  - Duty cycle
  - Addressing mode
  - Own Address 1
  - Dual Addressing mode
  - Own Address 2
  - General call mode
  - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx periperal.
- ***HAL_I2C_Init()***
- ***HAL_I2C_DeInit()***
- ***HAL_I2C_MspInit()***
- ***HAL_I2C_MspDeInit()***

21.2.3 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()

- ***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()*

21.2.4 Peripheral State and Errors functions

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

- *HAL_I2C_GetState()*
- *HAL_I2C_GetError()*

21.2.5 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 create the associated handle.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• HAL status

21.2.6 HAL_I2C_DeInit

Function Name	**HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)**
Function Description	DeInitializes the I2C peripheral.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• HAL status

21.2.7 HAL_I2C_MspInit

Function Name	**void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)**
Function Description	I2C MSP Init.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

21.2.8 HAL_I2C_MspDeInit

Function Name	**void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)**
Function Description	I2C MSP DeInit.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that

contains the configuration information for the specified I2C.

Return values • None

21.2.9 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

21.2.10 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

21.2.11 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

21.2.12 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

21.2.13 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 no-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

21.2.14 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 no-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

21.2.15 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 no-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

21.2.16 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 no-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

21.2.17 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 no-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

21.2.18 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 no-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

21.2.19 HAL_I2C_Slave_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function Description	Transmit in slave mode an amount of data in no-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

21.2.20 HAL_I2C_Slave_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)**
Function Description	Receive in slave mode an amount of data in no-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

21.2.21 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

21.2.22 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

21.2.23 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 no-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

21.2.24 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 no-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

21.2.25 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 no-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

21.2.26 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 no-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

21.2.27 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

21.2.28 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

21.2.29 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 I2C module
Return values	• HAL status

21.2.30 HAL_I2C_MasterTxCpltCallback

Function Name	**void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Master Tx Transfer completed callbacks.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

21.2.31 HAL_I2C_MasterRxCpltCallback

Function Name	**void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)**

Function Description	Master Rx Transfer completed callbacks.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

21.2.32 HAL_I2C_SlaveTxCpltCallback

Function Name	**void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Slave Tx Transfer completed callbacks.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

21.2.33 HAL_I2C_SlaveRxCpltCallback

Function Name	**void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Slave Rx Transfer completed callbacks.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

21.2.34 HAL_I2C_MemTxCpltCallback

Function Name	**void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Memory Tx Transfer completed callbacks.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

21.2.35 HAL_I2C_MemRxCpltCallback

Function Name	**void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Memory Rx Transfer completed callbacks.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

21.2.36 HAL_I2C_ErrorCallback

Function Name	**void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)**
Function Description	I2C error callbacks.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

21.2.37 HAL_I2C_GetState

Function Name	**HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)**
Function Description	Returns the I2C state.
Parameters	• **hi2c:** : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• HAL state

21.2.38 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

21.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

21.3.1 I2C

I2C

I2C addressing mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C dual addressing mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C Duty Cycle

I2C_DUTYCYCLE_2

I2C_DUTYCYCLE_16_9

I2C Error Codes

HAL_I2C_ERROR_NONE	No error
HAL_I2C_ERROR_BERR	BERR error
HAL_I2C_ERROR_ARLO	ARLO error
HAL_I2C_ERROR_AF	AF error
HAL_I2C_ERROR_OVR	OVR error
HAL_I2C_ERROR_DMA	DMA transfer error
HAL_I2C_ERROR_TIMEOUT	Timeout 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 interrupts.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  – I2C_IT_BUF: Buffer interrupt enable
  – I2C_IT_EVT: Event interrupt enable
  – I2C_IT_ERR: Error interrupt enable

Return value:

- None:

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupts.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  – I2C_IT_BUF: Buffer interrupt enable
  – I2C_IT_EVT: Event interrupt enable
  – I2C_IT_ERR: Error interrupt enable

Return value:

- None:

__HAL_I2C_GET_IT_SOURCE

Description:

- Checks if the specified I2C interrupt source is enabled or disabled.

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_BUF: Buffer interrupt enable
  - I2C_IT_EVT: Event interrupt enable
  - I2C_IT_ERR: Error interrupt enable

Return value:

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

__HAL_I2C_GET_FLAG

Description:

- Checks 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_SMBALERT: SMBus Alert flag
  - I2C_FLAG_TIMEOUT: Timeout or Tlow error flag
  - I2C_FLAG_PECERR: PEC error in reception flag
  - I2C_FLAG_OVR: Overrun/Underrun flag
  - I2C_FLAG_AF: Acknowledge failure flag
  - I2C_FLAG_ARLO: Arbitration lost flag
  - I2C_FLAG_BERR: Bus error flag
  - I2C_FLAG_TXE: Data register empty flag
  - I2C_FLAG_RXNE: Data register not empty flag
  - I2C_FLAG_STOPF: Stop detection flag
  - I2C_FLAG_ADD10: 10-bit header sent flag
  - I2C_FLAG_BTF: Byte transfer finished flag
  - I2C_FLAG_ADDR: Address sent flag Address matched flag
  - I2C_FLAG_SB: Start bit flag
  - I2C_FLAG_DUALF: Dual flag
  - I2C_FLAG_SMBHOST: SMBus host header
  - I2C_FLAG_SMBDEFAULT: SMBus default header

- I2C_FLAG_GENCALL: General call header flag
- I2C_FLAG_TRA: Transmitter/Receiver flag
- I2C_FLAG_BUSY: Bus busy flag
- I2C_FLAG_MSL: Master/Slave flag

Return value:

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

__HAL_I2C_CLEAR_FLAG

Description:

- Clears the I2C pending flags which are cleared by writing 0 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_SMBALERT: SMBus Alert flag
  - I2C_FLAG_TIMEOUT: Timeout or Tlow error flag
  - I2C_FLAG_PECERR: PEC error in reception flag
  - I2C_FLAG_OVR: Overrun/Underrun flag (Slave mode)
  - I2C_FLAG_AF: Acknowledge failure flag
  - I2C_FLAG_ARLO: Arbitration lost flag (Master mode)
  - I2C_FLAG_BERR: Bus error flag

Return value:

- None:

__HAL_I2C_CLEAR_ADDRFLAG

Description:

- Clears the I2C ADDR pending flag.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None:

__HAL_I2C_CLEAR_STOPFLAG

Description:

- Clears the I2C STOPF pending flag.

Parameters:

- __HANDLE__: specifies the I2C Handle.

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:

I2C Flag definition

I2C_FLAG_SMBALERT

I2C_FLAG_TIMEOUT

I2C_FLAG_PECERR

I2C_FLAG_OVR

I2C_FLAG_AF

I2C_FLAG_ARLO

I2C_FLAG_BERR

I2C_FLAG_TXE

I2C_FLAG_RXNE

I2C_FLAG_STOPF

I2C_FLAG_ADD10

I2C_FLAG_BTF

I2C_FLAG_ADDR

I2C_FLAG_SB

I2C_FLAG_DUALF

I2C_FLAG_SMBHOST

I2C_FLAG_SMBDEFAULT

I2C_FLAG_GENCALL

I2C_FLAG_TRA

I2C_FLAG_BUSY

I2C_FLAG_MSL

I2C_FLAG_MASK

I2C general call addressing mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_BUF

I2C_IT_EVT

I2C_IT_ERR

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C nostretch mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C Private Constants

I2C_TIMEOUT_FLAG

I2C_TIMEOUT_ADDR_SLAVE

I2C_STANDARD_MODE_MAX_CLK

I2C_FAST_MODE_MAX_CLK

I2C Private Macros

IS_I2C_ADDRESSING_MODE

IS_I2C_DUAL_ADDRESS

IS_I2C_GENERAL_CALL

IS_I2C_MEMADD_SIZE

IS_I2C_NO_STRETCH

IS_I2C_OWN_ADDRESS1

IS_I2C_OWN_ADDRESS2

IS_I2C_CLOCK_SPEED

IS_I2C_DUTY_CYCLE

I2C_FREQ_RANGE

I2C_RISE_TIME

I2C_SPEED_STANDARD

I2C_SPEED_FAST

I2C_SPEED

I2C_MEM_ADD_MSB

I2C_MEM_ADD_LSB

I2C_7BIT_ADD_WRITE

I2C_7BIT_ADD_READ

I2C_10BIT_ADDRESS

I2C_10BIT_HEADER_WRITE

I2C_10BIT_HEADER_READ

22 HAL I2S Generic Driver

22.1 I2S Firmware driver registers structures

22.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32f1xx_hal_i2s.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t CPOL***

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***

22.1.2 I2S_HandleTypeDef

I2S_HandleTypeDef is defined in the stm32f1xx_hal_i2s.h

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***
- ***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_InitTypeDef I2S_HandleTypeDef::Init***
- ***uint16_t* I2S_HandleTypeDef::pTxBuffPtr***
- ***__IO uint16_t I2S_HandleTypeDef::TxXferSize***
- ***__IO uint16_t I2S_HandleTypeDef::TxXferCount***
- ***uint16_t* I2S_HandleTypeDef::pRxBuffPtr***
- ***__IO uint16_t I2S_HandleTypeDef::RxXferSize***
- ***__IO uint16_t I2S_HandleTypeDef::RxXferCount***
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx***
- ***__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock***
- ***__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State***
- ***__IO uint32_t I2S_HandleTypeDef::ErrorCode***

22.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

22.2.1 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.
   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 initilalized 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. 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. The I2SxCLK source is the system clock (provided by the HSI, the HSE or the PLL, and sourcing the AHB clock). For connectivity line devices, the I2SxCLK source can be either SYSCLK or the PLL3 VCO (2 x PLL3CLK) clock in

order to achieve the maximum accuracy. Make sure that either: External clock source is configured after setting correctly the define constant HSE_VALUE in the stm32f1xx_hal_conf.h file.

4. 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 USART 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

You can refer to the I2S HAL driver header file for more useful macros

I2C Workarounds linked to Silicon Limitation

Only the 16-bit mode with no data extension can be used when the I2S is in Master and used the PCM long synchronization mode.

22.2.2 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
- Call the function HAL_I2S_DeInit() to restore the default configuration of the selected I2Sx periperal.
- ***HAL_I2S_Init()***
- ***HAL_I2S_DeInit()***
- ***HAL_I2S_MspInit()***
- ***HAL_I2S_MspDeInit()***

22.2.3 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_RxCpltCallback()
   - HAL_I2S_ErrorCallback()

- ***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()***

22.2.4 Peripheral State and Errors functions

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

- ***HAL_I2S_GetState()***
- ***HAL_I2S_GetError()***

22.2.5 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

22.2.6 HAL_I2S_DeInit

Function Name	**HAL_StatusTypeDef HAL_I2S_DeInit (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

22.2.7 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

22.2.8 HAL_I2S_MspDeInit

Function Name	**void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)**
Function Description	I2S MSP DeInit.
Parameters	• **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	• None

22.2.9 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 data sample to be sent: • **Timeout:** Timeout duration
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).

22.2.10 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 data sample to be sent: • **Timeout:** Timeout duration
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). • In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continouse way and as the I2S is not disabled at the end of the I2S transaction.

22.2.11 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).

22.2.12 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.

22.2.13 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).

22.2.14 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).

22.2.15 HAL_I2S_DMAPause

Function Name	**HAL_StatusTypeDef HAL_I2S_DMAPause (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

22.2.16 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

22.2.17 HAL_I2S_DMAStop

Function Name	**HAL_StatusTypeDef HAL_I2S_DMAStop (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

22.2.18 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

22.2.19 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

22.2.20 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

22.2.21 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

22.2.22 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

22.2.23 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

22.2.24 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

22.2.25 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

22.3 I2S Firmware driver defines

The following section lists the various define and macros of the module.

22.3.1 I2S

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

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error Codes

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

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_RXNE: Receive buffer not empty flag
  – I2S_FLAG_TXE: Transmit buffer empty flag
  – I2S_FLAG_UDR: Underrun flag
  – I2S_FLAG_OVR: Overrun flag
  – I2S_FLAG_CHSIDE: Channel Side flag
  – I2S_FLAG_BSY: Busy flag

Return value:

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

__HAL_I2S_CLEAR_OVRFLAG

Description:

- Clears the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None:

__HAL_I2S_CLEAR_UDRFLAG

Description:

- Clears the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None:

I2S Flag definition

I2S_FLAG_TXE

I2S_FLAG_RXNE

I2S_FLAG_UDR

I2S_FLAG_OVR

I2S_FLAG_FRE

I2S_FLAG_CHSIDE

I2S_FLAG_BSY

I2S Interrupt configuration definition

I2S_IT_TXE

I2S_IT_RXNE

I2S_IT_ERR

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 Private Macros

IS_I2S_MODE

IS_I2S_STANDARD

IS_I2S_DATA_FORMAT

IS_I2S_MCLK_OUTPUT

IS_I2S_AUDIO_FREQ

IS_I2S_CPOL

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

23 HAL IRDA Generic Driver

23.1 IRDA Firmware driver registers structures

23.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32f1xx_hal_irda.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint8_t Prescaler***
- ***uint32_t IrDAMode***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate*** This member configures the IRDA communication baud rate. The baud rate is computed using the following formula:
  – IntegerDivider = ((PCLKx) / (16 * (hirda->Init.BaudRate)))
  – FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***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 wether 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 prescaler value to be programmed in the IrDA low-power Baud Register, for defining pulse width on which burst acceptance/rejection will be decided. This value is used as divisor of system clock to achieve required pulse width.
- ***uint32_t IRDA_InitTypeDef::IrDAMode*** Specifies the IrDA mode This parameter can be a value of ***IRDA_Low_Power***

23.1.2 IRDA_HandleTypeDef

IRDA_HandleTypeDef is defined in the stm32f1xx_hal_irda.h

Data Fields

- ***USART_TypeDef * Instance***
- ***IRDA_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***

- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* IRDA_HandleTypeDef::Instance*** USART registers base address
- ***IRDA_InitTypeDef IRDA_HandleTypeDef::Init*** IRDA communication parameters
- ***uint8_t* IRDA_HandleTypeDef::pTxBuffPtr*** Pointer to IRDA Tx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::TxXferSize*** IRDA Tx Transfer size
- ***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
- ***uint16_t IRDA_HandleTypeDef::RxXferCount*** IRDA Rx Transfer Counter
- ***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::State*** IRDA communication state
- ***__IO uint32_t IRDA_HandleTypeDef::ErrorCode*** IRDA Error code

23.2 IRDA Firmware driver API description

The following section lists the various functions of the IRDA library.

23.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure.
2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API:
   a. Enable the USARTx interface clock.
   b. IRDA pins configuration:
      – Enable the clock for the IRDA GPIOs.
      – Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
   c. NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
      – Configure the USARTx interrupt priority.
      – Enable the NVIC USART IRQ handle.
   d. 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 initilalized 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.
- Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)

3. Program the Baud Rate, Word Length, Parity, IrDA Mode, Prescaler and Mode(Receiver/Transmitter) in the hirda 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 customed HAL_IRDA_MspInit() API. The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission 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 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 the specified IRDA interrupt has occurred or not

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

23.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in 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. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible IRDA frame formats are as listed in *Table 17: "IRDA frame formats"*
  - Prescaler: A pulse of width less than two and greater than one PSC period(s) may or may not be rejected. The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).
  - Mode: Receiver/transmitter modes
  - IrDAMode: the IrDA can operate in the Normal mode or in the Low power mode.

Table 17: IRDA frame formats

M bit	PCE bit	IRDA frame
0	0	| SB | 8 bit data | STB |
0	1	| SB | 7 bit data | PB | STB |
1	0	| SB | 9 bit data | STB |
1	1	| SB | 8 bit data | PB | STB |

The HAL_IRDA_Init() function follows IRDA configuration procedures (details for the procedures are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

- ***HAL_IRDA_Init()***
- ***HAL_IRDA_DeInit()***
- ***HAL_IRDA_MspInit()***
- ***HAL_IRDA_MspDeInit()***

23.2.3 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.
   - 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 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()

- ***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_IRQHandler()***
- ***HAL_IRDA_TxCpltCallback()***
- ***HAL_IRDA_TxHalfCpltCallback()***
- ***HAL_IRDA_RxCpltCallback()***
- ***HAL_IRDA_RxHalfCpltCallback()***
- ***HAL_IRDA_ErrorCallback()***

23.2.4 Peripheral State and Errors 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.
- HAL_IRDA_GetError() check in run-time errors that could be occurred during communication.
- ***HAL_IRDA_GetState()***
- ***HAL_IRDA_GetError()***

23.2.5 HAL_IRDA_Init

Function Name	**HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)**
Function Description	Initializes the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and create 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

23.2.6 HAL_IRDA_DeInit

Function Name	**HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)**
Function Description	DeInitializes 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

23.2.7 HAL_IRDA_MspInit

Function Name	**void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)**
Function Description	IRDA MSP Init.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None

23.2.8 HAL_IRDA_MspDeInit

Function Name	**void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)**
Function Description	IRDA MSP DeInit.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None

23.2.9 HAL_IRDA_Transmit

Function Name	**HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Sends 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

23.2.10 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

23.2.11 HAL_IRDA_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function Description	Sends an amount of data in non-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
Return values	• HAL status

23.2.12 HAL_IRDA_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function Description	Receives an amount of data in non-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
Return values	• HAL status

23.2.13 HAL_IRDA_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function Description	Sends an amount of data in non-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
Return values	• HAL status

23.2.14 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 non-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
Return values	• HAL status
Notes	• When the IRDA parity is enabled (PCE = 1) the data received

contain the parity bit.

23.2.15 HAL_IRDA_DMAPause

Function Name	**HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)**
Function Description	Pauses 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

23.2.16 HAL_IRDA_DMAResume

Function Name	**HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)**
Function Description	Resumes the DMA Transfer.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

23.2.17 HAL_IRDA_DMAStop

Function Name	**HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)**
Function Description	Stops the DMA Transfer.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

23.2.18 HAL_IRDA_IRQHandler

Function Name	**void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)**
Function Description	This function handles IRDA interrupt request.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None

23.2.19 HAL_IRDA_TxCpltCallback

Function Name	**void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)**
Function Description	Tx Transfer completed callbacks.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None

23.2.20 HAL_IRDA_TxHalfCpltCallback

Function Name	**void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**
Function Description	Tx Half Transfer completed callbacks.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

23.2.21 HAL_IRDA_RxCpltCallback

Function Name	**void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)**
Function Description	Rx Transfer completed callbacks.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None

23.2.22 HAL_IRDA_RxHalfCpltCallback

Function Name	**void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**
Function Description	Rx Half Transfer complete callbacks.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None

23.2.23 HAL_IRDA_ErrorCallback

Function Name	**void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)**
Function Description	IRDA error callbacks.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• None

23.2.24 HAL_IRDA_GetState

Function Name	**HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)**
Function Description	Returns the IRDA state.
Parameters	• **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	• HAL state

23.2.25 HAL_IRDA_GetError

Function Name	**uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)**
Function Description	Return the IRDA 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

23.3 IRDA Firmware driver defines

The following section lists the various define and macros of the module.

23.3.1 IRDA

IRDA

IRDA Error Codes

HAL_IRDA_ERROR_NONE	No error
HAL_IRDA_ERROR_PE	Parity error
HAL_IRDA_ERROR_NE	Noise error
HAL_IRDA_ERROR_FE	frame error
HAL_IRDA_ERROR_ORE	Overrun error
HAL_IRDA_ERROR_DMA	DMA transfer error

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE **Description:**

- Reset IRDA handle state.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_IRDA_FLUSH_DRREGISTER

Description:

- Flush the IRDA DR register.

Parameters:

- __HANDLE__: specifies the USART Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_IRDA_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - 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_IDLE: Idle Line detection 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_CLEAR_FLAG

Description:

- Clear the specified IRDA pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA_FLAG_TC: Transmission Complete flag.
  - IRDA_FLAG_RXNE: Receive data register not empty flag.

Return value:

- None:

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_IRDA_CLEAR_IDLEFLAG

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __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. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __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_SOURCE

Description:

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __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: Error interrupt
  - IRDA_IT_PE: Parity Error interrupt

Return value:

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

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

IRDA Flags

IRDA_FLAG_TXE

IRDA_FLAG_TC

IRDA_FLAG_RXNE

IRDA_FLAG_IDLE

IRDA_FLAG_ORE

IRDA_FLAG_NE

IRDA_FLAG_FE

IRDA_FLAG_PE

IRDA Interrupt Definitions

IRDA_IT_PE

IRDA_IT_TXE

IRDA_IT_TC

IRDA_IT_RXNE

IRDA_IT_IDLE

IRDA_IT_LBD

IRDA_IT_CTS

IRDA_IT_ERR

IRDA Low Power

IRDA_POWERMODE_LOWPOWER

IRDA_POWERMODE_NORMAL

IRDA Parity

IRDA_PARITY_NONE

IRDA_PARITY_EVEN

IRDA_PARITY_ODD

IRDA Private Constants

IRDA_DR_MASK_U16_8DATABITS

IRDA_DR_MASK_U16_9DATABITS

IRDA_DR_MASK_U8_7DATABITS

IRDA_DR_MASK_U8_8DATABITS

IRDA Private Macros

IRDA_CR1_REG_INDEX

IRDA_CR2_REG_INDEX

IRDA_CR3_REG_INDEX

IRDA_DIV

IRDA_DIVMANT

IRDA_DIVFRAQ

IRDA_BRR

IS_IRDA_BAUDRATE	The maximum Baud Rate is 115200bps Returns : True or False

IS_IRDA_WORD_LENGTH

IS_IRDA_PARITY

IS_IRDA_MODE

IS_IRDA_POWERMODE

IRDA_IT_MASK

IRDA Transfer Mode

IRDA_MODE_RX

IRDA_MODE_TX

IRDA_MODE_TX_RX

IRDA Word Length

IRDA_WORDLENGTH_8B

IRDA_WORDLENGTH_9B

24 HAL IWDG Generic Driver

24.1 IWDG Firmware driver registers structures

24.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32f1xx_hal_iwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler*** Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload*** Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF

24.1.2 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the stm32f1xx_hal_iwdg.h

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IWDG_StateTypeDef State***

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDef::Instance*** Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDef::Init*** IWDG required parameters
- ***HAL_LockTypeDef IWDG_HandleTypeDef::Lock*** IWDG Locking object
- ***__IO HAL_IWDG_StateTypeDef IWDG_HandleTypeDef::State*** IWDG communication state

24.2 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

24.2.1 IWDG specific features

- The IWDG can be started by either software or hardware (configurable through option byte).

- The IWDG is clocked by its own dedicated 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 cannot be disabled (LSI cannot be disabled too), and the counter starts counting down from the reset value of 0xFFF. When it reaches the end of count value (0x000) a system reset is generated.
- The IWDG counter should be refreshed at regular intervals, otherwise the watchdog generates an MCU reset when the counter reaches 0.
- 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.
- Min-max timeout value at 40KHz (LSI): 0.1us / 26.2s . The IWDG timeout may vary due to LSI frequency dispersion. STM32F1xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM5 CH4 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy. For more information, please refer to the STM32F1xx Reference manual. Note: LSI Calibration is only available on: High density, XL-density and Connectivity line devices.

24.2.2 How to use this driver

- Use IWDG using HAL_IWDG_Init() function to :
  - Enable write access to IWDG_PR, IWDG_RLR.
  - Configure the IWDG prescaler, counter reload value. This reload value will be loaded in the IWDG counter each time the counter is reloaded, then the IWDG will start counting down from this value.
- Use IWDG using HAL_IWDG_Start() function to :
  - Reload IWDG counter with value defined in the IWDG_RLR register.
  - Start the IWDG, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).
- 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
- __HAL_IWDG_GET_FLAG: Get the selected IWDG's flag status

24.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and create the associated handle
- Initialize the IWDG MSP
- DeInitialize IWDG MSP
- ***HAL_IWDG_Init()***

- ***HAL_IWDG_MspInit()***

24.2.4 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.
- ***HAL_IWDG_Start()***
- ***HAL_IWDG_Refresh()***

24.2.5 Peripheral State functions

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

- ***HAL_IWDG_GetState()***

24.2.6 HAL_IWDG_Init

Function Name	**HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)**
Function Description	Initializes the IWDG according to the specified parameters in the IWDG_InitTypeDef and creates the associated handle.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• HAL status

24.2.7 HAL_IWDG_MspInit

Function Name	**void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)**
Function Description	Initializes the IWDG MSP.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• None

24.2.8 HAL_IWDG_Start

Function Name	**HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)**
Function Description	Starts the IWDG.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values	• HAL status

24.2.9 HAL_IWDG_Refresh

Function Name	**HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)**
Function Description	Refreshes the IWDG.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• HAL status

24.2.10 HAL_IWDG_GetState

Function Name	**HAL_IWDG_StateTypeDef HAL_IWDG_GetState (IWDG_HandleTypeDef * hiwdg)**
Function Description	Returns the IWDG state.
Parameters	• **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	• HAL state

24.3 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

24.3.1 IWDG

IWDG

IWDG Exported Macros

__HAL_IWDG_RESET_HANDLE_STATE	**Description:** • Reset IWDG handle state. **Parameters:** • __HANDLE__: IWDG handle. **Return value:** • None:
__HAL_IWDG_START	**Description:** • Enables the IWDG peripheral. **Parameters:** • __HANDLE__: IWDG handle **Return value:**

- None:

__HAL_IWDG_RELOAD_COUNTER

Description:

- Reloads IWDG counter with value defined in the reload register (write access to IWDG_PR and IWDG_RLR registers disabled).

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None:

__HAL_IWDG_GET_FLAG

Description:

- Gets the selected IWDG's flag status.

Parameters:

- __HANDLE__: IWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IWDG_FLAG_PVU: Watchdog counter reload value update flag
  - IWDG_FLAG_RVU: Watchdog counter prescaler value flag

Return value:

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

IWDG Flag definition

IWDG_FLAG_PVU Watchdog counter prescaler value update Flag

IWDG_FLAG_RVU Watchdog counter reload value update Flag

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 Private Constants

IWDG_DEFAULT_TIMEOUT

IWDG Private Macros

IWDG_ENABLE_WRITE_ACCESS

Description:

- Enables write access to IWDG_PR and

IWDG_RLR registers.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None:

IWDG_DISABLE_WRITE_ACCESS

Description:

- Disables write access to IWDG_PR and IWDG_RLR registers.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None:

IS_IWDG_PRESCALER

IS_IWDG_RELOAD

IWDG Registers BitMask

IWDG_KR_KEY_RELOAD	IWDG Reload Counter Enable
IWDG_KR_KEY_ENABLE	IWDG Peripheral Enable
IWDG_KR_KEY_EWA	IWDG KR Write Access Enable
IWDG_KR_KEY_DWA	IWDG KR Write Access Disable

25 HAL NAND Generic Driver

25.1 NAND Firmware driver registers structures

25.1.1 NAND_IDTypeDef

NAND_IDTypeDef is defined in the stm32f1xx_hal_nand.h

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***

25.1.2 NAND_AddressTypedef

NAND_AddressTypedef is defined in the stm32f1xx_hal_nand.h

Data Fields

- ***uint16_t Page***
- ***uint16_t Zone***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypedef::Page*** NAND memory Page address
- ***uint16_t NAND_AddressTypedef::Zone*** NAND memory Zone address
- ***uint16_t NAND_AddressTypedef::Block*** NAND memory Block address

25.1.3 NAND_InfoTypeDef

NAND_InfoTypeDef is defined in the stm32f1xx_hal_nand.h

Data Fields

- ***uint32_t PageSize***
- ***uint32_t SpareAreaSize***
- ***uint32_t BlockSize***
- ***uint32_t BlockNbr***
- ***uint32_t ZoneSize***

Field Documentation

- ***uint32_t NAND_InfoTypeDef::PageSize*** NAND memory page (without spare area) size measured in K. bytes
- ***uint32_t NAND_InfoTypeDef::SpareAreaSize*** NAND memory spare area size measured in K. bytes
- ***uint32_t NAND_InfoTypeDef::BlockSize*** NAND memory block size number of pages
- ***uint32_t NAND_InfoTypeDef::BlockNbr*** NAND memory number of blocks
- ***uint32_t NAND_InfoTypeDef::ZoneSize*** NAND memory zone size measured in number of blocks

25.1.4 NAND_HandleTypeDef

NAND_HandleTypeDef is defined in the stm32f1xx_hal_nand.h

Data Fields

- ***FSMC_NAND_TypeDef * Instance***
- ***FSMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_InfoTypeDef Info***

Field Documentation

- ***FSMC_NAND_TypeDef* NAND_HandleTypeDef::Instance*** Register base address
- ***FSMC_NAND_InitTypeDef NAND_HandleTypeDef::Init*** NAND device control configuration parameters
- ***HAL_LockTypeDef NAND_HandleTypeDef::Lock*** NAND locking object
- ***__IO HAL_NAND_StateTypeDef NAND_HandleTypeDef::State*** NAND device access state
- ***NAND_InfoTypeDef NAND_HandleTypeDef::Info*** NAND characteristic information structure

25.2 NAND Firmware driver API description

The following section lists the various functions of the NAND library.

25.2.1 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 FSMC/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()/HAL_NAND_Read_SpareArea(), HAL_NAND_Write_Page()/HAL_NAND_Write_SpareArea() to read/write

page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the HAL_NAND_Info_TypeDef structure. The read/write address information is contained by the Nand_Address_Typedef structure passed as parameter.

- Perform NAND flash Reset chip operation using the function HAL_NAND_Reset().
- Perform NAND flash erase block operation using the function HAL_NAND_Erase_Block(). The erase block address information is contained in the Nand_Address_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL_NAND_Read_Status().
- You can also control the NAND device by calling the control APIs HAL_NAND_ECC_Enable()/ HAL_NAND_ECC_Disable() to respectively enable/disable the ECC code correction feature or the function HAL_NAND_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL_NAND_GetState()

This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

25.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

- ***HAL_NAND_Init()***
- ***HAL_NAND_DeInit()***
- ***HAL_NAND_MspInit()***
- ***HAL_NAND_MspDeInit()***
- ***HAL_NAND_IRQHandler()***
- ***HAL_NAND_ITCallback()***

25.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

- ***HAL_NAND_Read_ID()***
- ***HAL_NAND_Reset()***
- ***HAL_NAND_Read_Page()***
- ***HAL_NAND_Write_Page()***
- ***HAL_NAND_Read_SpareArea()***
- ***HAL_NAND_Write_SpareArea()***
- ***HAL_NAND_Erase_Block()***
- ***HAL_NAND_Read_Status()***
- ***HAL_NAND_Address_Inc()***

25.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

- ***HAL_NAND_ECC_Enable()***

- *HAL_NAND_ECC_Disable()*
- *HAL_NAND_GetECC()*

25.2.5 NAND State functions

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

- *HAL_NAND_GetState()*
- *HAL_NAND_Read_Status()*

25.2.6 HAL_NAND_Init

Function Name	**HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FSMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FSMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)**
Function Description	Perform NAND memory Initialization sequence.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **ComSpace_Timing:** pointer to Common space timing structure • **AttSpace_Timing:** pointer to Attribute space timing structure
Return values	• HAL status

25.2.7 HAL_NAND_DeInit

Function Name	**HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)**
Function Description	Perform NAND memory De-Initialization sequence.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• HAL status

25.2.8 HAL_NAND_MspInit

Function Name	**void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)**
Function Description	NAND MSP Init.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• None

25.2.9 HAL_NAND_MspDeInit

Function Name	**void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)**
Function Description	NAND MSP DeInit.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• None

25.2.10 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

25.2.11 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

25.2.12 HAL_NAND_Read_ID

Function Name	**HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)**
Function Description	Read the NAND memory electronic signature.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pNAND_ID:** NAND ID structure
Return values	• HAL status

25.2.13 HAL_NAND_Reset

Function Name	**HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)**
Function Description	NAND memory reset.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values	• HAL status

25.2.14 HAL_NAND_Read_Page

Function Name	**HAL_StatusTypeDef HAL_NAND_Read_Page (NAND_HandleTypeDef * hnand, NAND_AddressTypedef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)**
Function Description	Read Page(s) from NAND memory block.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pAddress:** : pointer to NAND address structure • **pBuffer:** : pointer to destination read buffer • **NumPageToRead:** : number of pages to read from block
Return values	• HAL status

25.2.15 HAL_NAND_Write_Page

Function Name	**HAL_StatusTypeDef HAL_NAND_Write_Page (NAND_HandleTypeDef * hnand, NAND_AddressTypedef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)**
Function Description	Write Page(s) to NAND memory block.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pAddress:** : pointer to NAND address structure • **pBuffer:** : pointer to source buffer to write • **NumPageToWrite:** : number of pages to write to block
Return values	• HAL status

25.2.16 HAL_NAND_Read_SpareArea

Function Name	**HAL_StatusTypeDef HAL_NAND_Read_SpareArea (NAND_HandleTypeDef * hnand, NAND_AddressTypedef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)**
Function Description	Read Spare area(s) from NAND memory.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pAddress:** : pointer to NAND address structure • **pBuffer:** pointer to source buffer to write • **NumSpareAreaToRead:** Number of spare area to read
Return values	• HAL status

25.2.17 HAL_NAND_Write_SpareArea

Function Name	**HAL_StatusTypeDef HAL_NAND_Write_SpareArea (NAND_HandleTypeDef * hnand, NAND_AddressTypedef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)**
Function Description	Write Spare area(s) to NAND memory.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **pAddress:** : pointer to NAND address structure • **pBuffer:** : pointer to source buffer to write • **NumSpareAreaTowrite:** : number of spare areas to write to block
Return values	• HAL status

25.2.18 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

25.2.19 HAL_NAND_Read_Status

Function Name	**uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)**
Function Description	NAND memory read status.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• NAND status

25.2.20 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 addressNAND_INVALID_ADDRESS: When the new address

is invalid address

25.2.21 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

25.2.22 HAL_NAND_ECC_Disable

Function Name	**HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)**
Function Description	Disables dynamically FSMC_NAND ECC feature.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• HAL status

25.2.23 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

25.2.24 HAL_NAND_GetState

Function Name	**HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hnand)**
Function Description	return the NAND state
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• HAL state

25.2.25 HAL_NAND_Read_Status

Function Name	**uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnand)**
Function Description	NAND memory read status.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• NAND status

25.3 NAND Firmware driver defines

The following section lists the various define and macros of the module.

25.3.1 NAND

NAND

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE

Description:

- Reset NAND handle state.

Parameters:

- __HANDLE__: specifies the NAND handle.

Return value:

- None:

NAND Private Constants

NAND_DEVICE1

NAND_DEVICE2

NAND_WRITE_TIMEOUT

CMD_AREA

ADDR_AREA

NAND_CMD_AREA_A

NAND_CMD_AREA_B

NAND_CMD_AREA_C

NAND_CMD_AREA_TRUE1

NAND_CMD_WRITE0

NAND_CMD_WRITE_TRUE1

NAND_CMD_ERASE0

NAND_CMD_ERASE1

NAND_CMD_READID

NAND_CMD_STATUS

NAND_CMD_LOCK_STATUS

NAND_CMD_RESET

NAND_VALID_ADDRESS

NAND_INVALID_ADDRESS

NAND_TIMEOUT_ERROR

NAND_BUSY

NAND_ERROR

NAND_READY

NAND Private Macros

__ARRAY_ADDRESS

Description:

- NAND memory address computation.

Parameters:

- __ADDRESS__: NAND memory address.
- __HANDLE__: : NAND handle.

Return value:

- NAND: Raw address value

__ADDR_1st_CYCLE

Description:

- NAND memory address cycling.

Parameters:

- __ADDRESS__: NAND memory address.

Return value:

- NAND: address cycling value.

__ADDR_2nd_CYCLE

__ADDR_3rd_CYCLE

__ADDR_4th_CYCLE

26 HAL NOR Generic Driver

26.1 NOR Firmware driver registers structures

26.1.1 NOR_IDTypeDef

NOR_IDTypeDef is defined in the stm32f1xx_hal_nor.h

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

26.1.2 NOR_CFITypeDef

NOR_CFITypeDef is defined in the stm32f1xx_hal_nor.h

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***

26.1.3 NOR_HandleTypeDef

NOR_HandleTypeDef is defined in the stm32f1xx_hal_nor.h

Data Fields

- ***FSMC_NORSRAM_TypeDef * Instance***
- ***FSMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FSMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***

Field Documentation

- ***FSMC_NORSRAM_TypeDef* NOR_HandleTypeDef::Instance*** Register base address
- ***FSMC_NORSRAM_EXTENDED_TypeDef* NOR_HandleTypeDef::Extended*** Extended mode register base address
- ***FSMC_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

26.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

26.2.1 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 FSMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()

This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- __NOR_WRITE : NOR memory write data to specified address

26.2.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

- *HAL_NOR_Init()*
- *HAL_NOR_DeInit()*
- *HAL_NOR_MspInit()*
- *HAL_NOR_MspDeInit()*
- *HAL_NOR_MspWait()*

26.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

- *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()*

26.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

- *HAL_NOR_WriteOperation_Enable()*
- *HAL_NOR_WriteOperation_Disable()*

26.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

- *HAL_NOR_GetState()*
- *HAL_NOR_GetStatus()*

26.2.6 HAL_NOR_Init

Function Name	**HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_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

26.2.7 HAL_NOR_DeInit

Function Name	**HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)**
Function Description	Perform NOR memory De-Initialization sequence.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	• HAL status

26.2.8 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

26.2.9 HAL_NOR_MspDeInit

Function Name	**void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)**
Function Description	NOR MSP DeInit.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	• None

26.2.10 HAL_NOR_MspWait

Function Name	**void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)**
Function Description	NOR BSP Wait fro 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

26.2.11 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

26.2.12 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

26.2.13 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

26.2.14 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

26.2.15 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 block of data from the FSMC NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **uwAddress:** NOR memory internal address to read from. • **pData:** pointer to the buffer that receives the data read from the NOR memory. • **uwBufferSize:** : number of Half word to read.
Return values	• HAL status

26.2.16 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 FSMC NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **uwAddress:** NOR memory internal address from which the data • **pData:** pointer to source data buffer. • **uwBufferSize:** number of Half words to write.
Return values	• HAL status
Notes	• Some NOR memory need Address aligned to xx bytes (can be aligned to 64 bytes boundary for example). • The maximum buffer size allowed is NOR memory dependent (can be 64 Bytes max for example).

26.2.17 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

26.2.18 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

26.2.19 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

26.2.20 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

26.2.21 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

26.2.22 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

26.2.23 HAL_NOR_GetStatus

Function Name	**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 progamming Timeout
Return values	• NOR_Status The returned value can be: NOR_SUCCESS, NOR_ERROR or NOR_TIMEOUT

26.3 NOR Firmware driver defines

The following section lists the various define and macros of the module.

26.3.1 NOR

NOR

NOR Exported Macros

__HAL_NOR_RESET_HANDLE_STATE

Description:

- Reset NOR handle state.

Parameters:

- __HANDLE__: NOR handle

Return value:

- None:

NOR Private Constants

NOR_CMD_ADDRESS_FIRST

NOR_CMD_ADDRESS_FIRST_CFI

NOR_CMD_ADDRESS_SECOND

NOR_CMD_ADDRESS_THIRD

NOR_CMD_ADDRESS_FOURTH

NOR_CMD_ADDRESS_FIFTH

NOR_CMD_ADDRESS_SIXTH

NOR_CMD_DATA_READ_RESET

NOR_CMD_DATA_FIRST

NOR_CMD_DATA_SECOND

NOR_CMD_DATA_AUTO_SELECT

NOR_CMD_DATA_PROGRAM

NOR_CMD_DATA_CHIP_BLOCK_ERASE_THIRD

NOR_CMD_DATA_CHIP_BLOCK_ERASE_FOURTH

NOR_CMD_DATA_CHIP_BLOCK_ERASE_FIFTH

NOR_CMD_DATA_CHIP_ERASE

NOR_CMD_DATA_CFI

NOR_CMD_DATA_BUFFER_AND_PROG

NOR_CMD_DATA_BUFFER_AND_PROG_CONFIRM

NOR_CMD_DATA_BLOCK_ERASE

NOR_MASK_STATUS_DQ5

NOR_MASK_STATUS_DQ6

MC_ADDRESS

DEVICE_CODE1_ADDR

DEVICE_CODE2_ADDR

DEVICE_CODE3_ADDR

CFI1_ADDRESS

CFI2_ADDRESS

CFI3_ADDRESS

CFI4_ADDRESS

NOR_TMEOUT

NOR_MEMORY_8B

NOR_MEMORY_16B

NOR_MEMORY_ADRESS1

NOR_MEMORY_ADRESS2

NOR_MEMORY_ADRESS3

NOR_MEMORY_ADRESS4

NOR Private Macros

__NOR_ADDR_SHIFT

Description:

- NOR memory address shifting.

Parameters:

- __NOR_ADDRESS: NOR base address

- __NOR_MEMORY_WIDTH_: NOR memory width
- __ADDRESS__: NOR memory address

Return value:

- NOR: shifted address value

__NOR_WRITE

Description:

- NOR memory write data to specified address.

Parameters:

- __ADDRESS__: NOR memory address
- __DATA__: Data to write

Return value:

- None:

27 HAL PCCARD Generic Driver

27.1 PCCARD Firmware driver registers structures

27.1.1 PCCARD_HandleTypeDef

PCCARD_HandleTypeDef is defined in the stm32f1xx_hal_pccard.h

Data Fields

- ***FSMC_PCCARD_TypeDef * Instance***
- ***FSMC_PCCARD_InitTypeDef Init***
- ***__IO HAL_PCCARD_StateTypeDef State***
- ***HAL_LockTypeDef Lock***

Field Documentation

- ***FSMC_PCCARD_TypeDef* PCCARD_HandleTypeDef::Instance*** Register base address for PCCARD device
- ***FSMC_PCCARD_InitTypeDef PCCARD_HandleTypeDef::Init*** PCCARD device control configuration parameters
- ***__IO HAL_PCCARD_StateTypeDef PCCARD_HandleTypeDef::State*** PCCARD device access state
- ***HAL_LockTypeDef PCCARD_HandleTypeDef::Lock*** PCCARD Lock

27.2 PCCARD Firmware driver API description

The following section lists the various functions of the PCCARD library.

27.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control PCCARD/compact flash memories. It uses the FSMC/FSMC layer functions to interface with PCCARD devices. This driver is used for:

- PCCARD/compact flash memory configuration sequence using the function HAL_PCCARD_Init() with control and timing parameters for both common and attribute spaces.
- Read PCCARD/compact flash memory maker and device IDs using the function HAL_CF_Read_ID(). The read information is stored in the CompactFlash_ID structure declared by the function caller.
- Access PCCARD/compact flash memory by read/write operations using the functions HAL_CF_Read_Sector()/HAL_CF_Write_Sector(), to read/write sector.
- Perform PCCARD/compact flash Reset chip operation using the function HAL_CF_Reset().
- Perform PCCARD/compact flash erase sector operation using the function HAL_CF_Erase_Sector().
- Read the PCCARD/compact flash status operation using the function HAL_CF_ReadStatus().

- You can monitor the PCCARD/compact flash device HAL state by calling the function HAL_PCCARD_GetState()

This driver is a set of generic APIs which handle standard PCCARD/compact flash operations. If a PCCARD/compact flash device contains different operations and/or implementations, it should be implemented separately.

27.2.2 PCCARD Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the PCCARD memory

- ***HAL_PCCARD_Init()***
- ***HAL_PCCARD_DeInit()***
- ***HAL_PCCARD_MspInit()***
- ***HAL_PCCARD_MspDeInit()***

27.2.3 PCCARD Input Output and memory functions

This section provides functions allowing to use and control the PCCARD memory

- ***HAL_CF_Read_ID()***
- ***HAL_CF_Read_Sector()***
- ***HAL_CF_Write_Sector()***
- ***HAL_CF_Erase_Sector()***
- ***HAL_CF_Reset()***
- ***HAL_PCCARD_IRQHandler()***
- ***HAL_PCCARD_ITCallback()***

27.2.4 PCCARD Peripheral State functions

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

- ***HAL_PCCARD_GetState()***
- ***HAL_CF_GetStatus()***
- ***HAL_CF_ReadStatus()***

27.2.5 HAL_PCCARD_Init

Function Name	**HAL_StatusTypeDef HAL_PCCARD_Init (PCCARD_HandleTypeDef * hpccard, FSMC_NAND_PCC_TimingTypeDef * ComSpaceTiming, FSMC_NAND_PCC_TimingTypeDef * AttSpaceTiming, FSMC_NAND_PCC_TimingTypeDef * IOSpaceTiming)**
Function Description	Perform the PCCARD memory Initialization sequence.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module. • **ComSpaceTiming:** Common space timing structure

	• **AttSpaceTiming:** Attribute space timing structure • **IOSpaceTiming:** IO space timing structure
Return values	• HAL status

27.2.6 HAL_PCCARD_DeInit

Function Name	**HAL_StatusTypeDef HAL_PCCARD_DeInit (PCCARD_HandleTypeDef * hpccard)**
Function Description	Perform the PCCARD memory De-initialization sequence.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• HAL status

27.2.7 HAL_PCCARD_MspInit

Function Name	**void HAL_PCCARD_MspInit (PCCARD_HandleTypeDef * hpccard)**
Function Description	PCCARD MSP Init.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• None

27.2.8 HAL_PCCARD_MspDeInit

Function Name	**void HAL_PCCARD_MspDeInit (PCCARD_HandleTypeDef * hpccard)**
Function Description	PCCARD MSP DeInit.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• None

27.2.9 HAL_CF_Read_ID

Function Name	**HAL_StatusTypeDef HAL_CF_Read_ID (PCCARD_HandleTypeDef * hpccard, uint8_t CompactFlash_ID, uint8_t * pStatus)**
Function Description	Read Compact Flash's ID.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD

module.
- **CompactFlash_ID:** Compact flash ID structure.
- **pStatus:** pointer to compact flash status

Return values	• HAL status

27.2.10 HAL_CF_Read_Sector

Function Name	**HAL_StatusTypeDef HAL_CF_Read_Sector (PCCARD_HandleTypeDef * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)**
Function Description	Read sector from PCCARD memory.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module. • **pBuffer:** pointer to destination read buffer • **SectorAddress:** Sector address to read • **pStatus:** pointer to CF status
Return values	• HAL status

27.2.11 HAL_CF_Write_Sector

Function Name	**HAL_StatusTypeDef HAL_CF_Write_Sector (PCCARD_HandleTypeDef * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)**
Function Description	Write sector to PCCARD memory.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module. • **pBuffer:** pointer to source write buffer • **SectorAddress:** Sector address to write • **pStatus:** pointer to CF status
Return values	• HAL status

27.2.12 HAL_CF_Erase_Sector

Function Name	**HAL_StatusTypeDef HAL_CF_Erase_Sector (PCCARD_HandleTypeDef * hpccard, uint16_t SectorAddress, uint8_t * pStatus)**
Function Description	Erase sector from PCCARD memory.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module. • **SectorAddress:** Sector address to erase • **pStatus:** pointer to CF status

Return values	• HAL status

27.2.13 HAL_CF_Reset

Function Name	**HAL_StatusTypeDef HAL_CF_Reset (PCCARD_HandleTypeDef * hpccard)**
Function Description	Reset the PCCARD memory.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• HAL status

27.2.14 HAL_PCCARD_IRQHandler

Function Name	**void HAL_PCCARD_IRQHandler (PCCARD_HandleTypeDef * hpccard)**
Function Description	This function handles PCCARD device interrupt request.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• HAL status

27.2.15 HAL_PCCARD_ITCallback

Function Name	**void HAL_PCCARD_ITCallback (PCCARD_HandleTypeDef * hpccard)**
Function Description	PCCARD interrupt feature callback.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• None

27.2.16 HAL_PCCARD_GetState

Function Name	**HAL_PCCARD_StateTypeDef HAL_PCCARD_GetState (PCCARD_HandleTypeDef * hpccard)**
Function Description	return the PCCARD controller state
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• HAL state

27.2.17 HAL_CF_GetStatus

Function Name	**CF_StatusTypedef HAL_CF_GetStatus (PCCARD_HandleTypeDef * hpccard)**
Function Description	Get the compact flash memory status.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• New status of the CF operation. This parameter can be: CompactFlash_TIMEOUT_ERROR: when the previous operation generate a Timeout errorCompactFlash_READY: when memory is ready for the next operation

27.2.18 HAL_CF_ReadStatus

Function Name	**CF_StatusTypedef HAL_CF_ReadStatus (PCCARD_HandleTypeDef * hpccard)**
Function Description	Reads the Compact Flash memory status using the Read status command.
Parameters	• **hpccard:** pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.
Return values	• The status of the Compact Flash memory. This parameter can be: CompactFlash_BUSY: when memory is busyCompactFlash_READY: when memory is ready for the next operationCompactFlash_ERROR: when the previous operation gererates error

27.3 PCCARD Firmware driver defines

The following section lists the various define and macros of the module.

27.3.1 PCCARD

PCCARD

PCCARD Exported Macros

__HAL_PCCARD_RESET_HANDLE_STATE	**Description:** • Reset PCCARD handle state. **Parameters:** • __HANDLE__: specifies the PCCARD handle. **Return value:** • None:

PCCARD Private Constants

PCCARD_TIMEOUT_READ_ID

PCCARD_TIMEOUT_SECTOR

PCCARD_TIMEOUT_STATUS

PCCARD_STATUS_OK

PCCARD_STATUS_WRITE_OK

CF_DEVICE_ADDRESS

CF_ATTRIBUTE_SPACE_ADDRESS

CF_COMMON_SPACE_ADDRESS

CF_IO_SPACE_ADDRESS

CF_IO_SPACE_PRIMARY_ADDR

CF_DATA

CF_SECTOR_COUNT

CF_SECTOR_NUMBER

CF_CYLINDER_LOW

CF_CYLINDER_HIGH

CF_CARD_HEAD

CF_STATUS_CMD

CF_STATUS_CMD_ALTERNATE

CF_COMMON_DATA_AREA

CF_CARD_CONFIGURATION

CF_READ_SECTOR_CMD

CF_WRITE_SECTOR_CMD

CF_ERASE_SECTOR_CMD

CF_IDENTIFY_CMD

CF_TIMEOUT_ERROR

CF_BUSY

CF_PROGR

CF_READY

CF_SECTOR_SIZE

28 HAL PCD Generic Driver

28.1 PCD Firmware driver registers structures

28.1.1 PCD_HandleTypeDef

PCD_HandleTypeDef is defined in the stm32f1xx_hal_pcd.h

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***uint32_t Setup***
- ***void * pData***

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance*** Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init*** PCD required parameters
- ***__IO uint8_t PCD_HandleTypeDef::USB_Address*** USB Address: not used by USB OTG FS
- ***PCD_EPTypeDef PCD_HandleTypeDef::IN_ep[15]*** IN endpoint parameters
- ***PCD_EPTypeDef PCD_HandleTypeDef::OUT_ep[15]*** OUT endpoint parameters
- ***HAL_LockTypeDef PCD_HandleTypeDef::Lock*** PCD peripheral status
- ***__IO PCD_StateTypeDef PCD_HandleTypeDef::State*** PCD communication state
- ***uint32_t PCD_HandleTypeDef::Setup[12]*** Setup packet buffer
- ***void* PCD_HandleTypeDef::pData*** Pointer to upper stack Handler

28.2 PCD Firmware driver API description

The following section lists the various functions of the PCD library.

28.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the HCD 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 the following macro
      – __HAL_RCC_USB_CLK_ENABLE(); For USB Device FS peripheral available on STM32F102xx and STM32F103xx devices

- __HAL_RCC_OTGFS_CLK_ENABLE(); For USB OTG FS peripheral available on STM32F105xx and STM32F107xx devices

   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 HCD transmission and reception:
   a. HAL_PCD_Start();

28.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- ***HAL_PCD_Init()***
- ***HAL_PCD_DeInit()***
- ***HAL_PCD_MspInit()***
- ***HAL_PCD_MspDeInit()***

28.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.

- ***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()***

28.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

- ***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_ActiveRemoteWakeup()*
- *HAL_PCD_DeActiveRemoteWakeup()*

28.2.5 Peripheral State functions

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

- *HAL_PCD_GetState()*

28.2.6 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

28.2.7 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

28.2.8 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

28.2.9 HAL_PCD_MspDeInit

Function Name	**void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)**
Function Description	DeInitializes PCD MSP.
Parameters	• **hpcd:** PCD handle
Return values	• None

28.2.10 HAL_PCD_Start

Function Name	**HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)**
Function Description	Start The USB Device.
Parameters	• **hpcd:** PCD handle
Return values	• HAL status

28.2.11 HAL_PCD_Stop

Function Name	**HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)**
Function Description	Stop The USB Device.
Parameters	• **hpcd:** PCD handle
Return values	• HAL status

28.2.12 HAL_PCD_IRQHandler

Function Name	**void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)**
Function Description	This function handles PCD interrupt request.
Parameters	• **hpcd:** PCD handle
Return values	• HAL status

28.2.13 HAL_PCD_DataOutStageCallback

Function Name	**void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function Description	Data out stage callbacks.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• None

28.2.14 HAL_PCD_DataInStageCallback

Function Name	**void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function Description	Data IN stage callbacks.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number

Return values	• None

28.2.15 HAL_PCD_SetupStageCallback

Function Name	**void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Setup stage callback.
Parameters	• **hpcd:** PCD handle
Return values	• None

28.2.16 HAL_PCD_SOFCallback

Function Name	**void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)**
Function Description	USB Start Of Frame callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

28.2.17 HAL_PCD_ResetCallback

Function Name	**void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)**
Function Description	USB Reset callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

28.2.18 HAL_PCD_SuspendCallback

Function Name	**void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Suspend event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

28.2.19 HAL_PCD_ResumeCallback

Function Name	**void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Resume event callbacks.
Parameters	• **hpcd:** PCD handle

Return values	• None

28.2.20 HAL_PCD_ISOOUTIncompleteCallback

Function Name	**void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function Description	Incomplete ISO OUT callbacks.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• None

28.2.21 HAL_PCD_ISOINIncompleteCallback

Function Name	**void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**
Function Description	Incomplete ISO IN callbacks.
Parameters	• **hpcd:** PCD handle • **epnum:** endpoint number
Return values	• None

28.2.22 HAL_PCD_ConnectCallback

Function Name	**void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Connection event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

28.2.23 HAL_PCD_DisconnectCallback

Function Name	**void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Disconnection event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

28.2.24 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

28.2.25 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

28.2.26 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

28.2.27 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

28.2.28 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

28.2.29 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

28.2.30 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

28.2.31 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

28.2.32 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

28.2.33 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

28.2.34 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

28.2.35 HAL_PCD_ActiveRemoteWakeup

Function Name	**HAL_StatusTypeDef HAL_PCD_ActiveRemoteWakeup (PCD_HandleTypeDef * hpcd)**
Function Description	HAL_PCD_ActiveRemoteWakeup : active remote wakeup signalling.
Parameters	• **hpcd:** PCD handle
Return values	• HAL status

28.2.36 HAL_PCD_DeActiveRemoteWakeup

Function Name	**HAL_StatusTypeDef HAL_PCD_DeActiveRemoteWakeup (PCD_HandleTypeDef * hpcd)**
Function Description	HAL_PCD_DeActiveRemoteWakeup : de-active remote wakeup signalling.
Parameters	• **hpcd:** PCD handle
Return values	• HAL status

28.2.37 HAL_PCD_GetState

Function Name	**PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)**
Function Description	Return the PCD state.
Parameters	• **hpcd:** PCD handle
Return values	• HAL state

28.3 PCD Firmware driver defines

The following section lists the various define and macros of the module.

28.3.1 PCD

PCD

PCD ENDP

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD Endpoint Kind

PCD_SNG_BUF

PCD_DBL_BUF

PCD EP0 MPS

PCD_EP0MPS_64

PCD_EP0MPS_32

PCD_EP0MPS_16

PCD_EP0MPS_08

PCD Exported Macros

__HAL_PCD_ENABLE

__HAL_PCD_DISABLE

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_WAKEUP_EXTI_DISABLE_IT

__HAL_USB_WAKEUP_EXTI_GET_FLAG

__HAL_USB_WAKEUP_EXTI_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_RISING_EDGE

__HAL_USB_WAKEUP_EXTI_ENABLE_FALLING_EDGE

__HAL_USB_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

PCD Instance definition

IS_PCD_ALL_INSTANCE

PCD PHY Module

PCD_PHY_EMBEDDED

PCD Private Macros

PCD_MIN

PCD_MAX

PCD_SET_ENDPOINT

PCD_GET_ENDPOINT

USB_EP0StartXfer

PCD_SET_EPTYPE

Description:

- sets the type in the endpoint register(bits EP_TYPE[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wType: Endpoint Type.

Return value:

- None:

PCD_GET_EPTYPE

Description:

- gets the type in the endpoint register(bits EP_TYPE[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- Endpoint: Type

PCD_FreeUserBuffer

Description:

- free buffer used from the application realizing it to the line toggles bit SW_BUF in the double buffered endpoint register

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

- bDir: Direction

Return value:

- None:

PCD_GET_DB_DIR

Description:

- gets direction of the double buffered endpoint

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- EP_DBUF_OUT: if the endpoint counter not yet programmed.

PCD_SET_EP_TX_STATUS

Description:

- sets the status for tx transfer (bits STAT_TX[1:0]).

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wState: new state

Return value:

- None:

PCD_SET_EP_RX_STATUS

Description:

- sets the status for rx transfer (bits STAT_TX[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wState: new state

Return value:

- None:

PCD_SET_EP_TXRX_STATUS

Description:

- sets the status for rx & tx (bits STAT_TX[1:0] & STAT_RX[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wStaterx: new state.
- wStatetx: new state.

Return value:

- None:

PCD_GET_EP_TX_STATUS

Description:

- gets the status for tx/rx transfer (bits STAT_TX[1:0] /STAT_RX[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- status:

PCD_GET_EP_RX_STATUS

PCD_SET_EP_TX_VALID

Description:

- sets directly the VALID tx/rx-status into the endpoint register

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_SET_EP_RX_VALID

PCD_GET_EP_TX_STALL_STATUS

Description:

- checks stall condition in an endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- TRUE: = endpoint in stall condition.

PCD_GET_EP_RX_STALL_STATUS

PCD_SET_EP_KIND

Description:

- set & clear EP_KIND bit.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_CLEAR_EP_KIND

PCD_SET_OUT_STATUS

Description:

- Sets/clears directly STATUS_OUT bit in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_CLEAR_OUT_STATUS

PCD_SET_EP_DBUF

Description:

- Sets/clears directly EP_KIND bit in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_CLEAR_EP_DBUF

PCD_CLEAR_RX_EP_CTR

Description:

- Clears bit CTR_RX / CTR_TX in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_CLEAR_TX_EP_CTR

PCD_RX_DTOG

Description:

- Toggles DTOG_RX / DTOG_TX bit in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_TX_DTOG

PCD_CLEAR_RX_DTOG

Description:

- Clears DTOG_RX / DTOG_TX bit in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_CLEAR_TX_DTOG

PCD_SET_EP_ADDRESS

Description:

- Sets address in an endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- bAddr: Address.

Return value:

- None:

PCD_GET_EP_ADDRESS

PCD_EP_TX_ADDRESS

PCD_EP_TX_CNT

PCD_EP_RX_ADDRESS

PCD_EP_RX_CNT

PCD_SET_EP_RX_CNT

PCD_SET_EP_TX_ADDRESS

Description:

- sets address of the tx/rx buffer.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wAddr: address to be set (must be word aligned).

Return value:

- None:

PCD_SET_EP_RX_ADDRESS

PCD_GET_EP_TX_ADDRESS

Description:

- Gets address of the tx/rx buffer.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- address: of the buffer.

PCD_GET_EP_RX_ADDRESS

PCD_CALC_BLK32

Description:

- Sets counter of rx buffer with no.

Parameters:

- dwReg: Register
- wCount: Counter.
- wNBlocks: no. of Blocks.

Return value:

- None:

PCD_CALC_BLK2

PCD_SET_EP_CNT_RX_REG

PCD_SET_EP_RX_DBUF0_CNT

PCD_SET_EP_TX_CNT

Description:

- sets counter for the tx/rx buffer.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wCount: Counter value.

Return value:

- None:

PCD_GET_EP_TX_CNT

Description:

- gets counter of the tx buffer.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- Counter: value

PCD_GET_EP_RX_CNT

PCD_SET_EP_DBUF0_ADDR

Description:

- Sets buffer 0/1 address in a double buffer endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.

Return value:

- Counter: value

PCD_SET_EP_DBUF1_ADDR

PCD_SET_EP_DBUF_ADDR

Description:

- Sets addresses in a double buffer endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.
- wBuf1Addr: = buffer 1 address.

Return value:

- None:

PCD_GET_EP_DBUF0_ADDR

Description:

- Gets buffer 0/1 address of a double buffer endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_GET_EP_DBUF1_ADDR

PCD_SET_EP_DBUF0_CNT

Description:

- Gets buffer 0/1 address of a double buffer endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- bDir: endpoint dir EP_DBUF_OUT = OUT EP_DBUF_IN = IN
- wCount: Counter value

Return value:

- None:

PCD_SET_EP_DBUF1_CNT

PCD_SET_EP_DBUF_CNT

PCD_GET_EP_DBUF0_CNT

Description:

- Gets buffer 0/1 rx/tx counter for double buffering.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_GET_EP_DBUF1_CNT

PCD Speed

PCD_SPEED_HIGH

PCD_SPEED_HIGH_IN_FULL

PCD_SPEED_FULL

29 HAL PCD Extension Driver

29.1 PCDEx Firmware driver API description

The following section lists the various functions of the PCDEx library.

29.1.1 Extended Peripheral Control functions

This section provides functions allowing to:

- Update FIFO (USB_OTG_FS)
- Update PMA configuration (USB)
- *HAL_PCDEx_PMAConfig()*

29.1.2 HAL_PCDEx_PMAConfig

Function Name	**HAL_StatusTypeDef HAL_PCDEx_PMAConfig (PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t ep_kind, uint32_t pmaadress)**
Function Description	Configure PMA for EP.
Parameters	• **hpcd:** : Device instance • **ep_addr:** endpoint address • **ep_kind:** endpoint Kind USB_SNG_BUF: Single Buffer used USB_DBL_BUF: Double Buffer used • **pmaadress:** EP address in The PMA: In case of single buffer endpoint this parameter is 16-bit value providing the address in PMA allocated to endpoint. In case of double buffer endpoint this parameter is a 32-bit value providing the endpoint buffer 0 address in the LSB part of 32-bit value and endpoint buffer 1 address in the MSB part of 32-bit value.
Return values	• HAL status

29.1.3 HAL_PCDEx_SetConnectionState

Function Name	**void HAL_PCDEx_SetConnectionState (PCD_HandleTypeDef * hpcd, uint8_t state)**
Function Description	Software Device Connection, this function is not required by USB OTG FS peripheral, it is used only by USB Device FS peripheral.
Parameters	• **hpcd:** PCD handle • **state:** connection state (0 : disconnected / 1: connected)
Return values	• None

29.2 PCDEx Firmware driver defines

The following section lists the various define and macros of the module.

29.2.1 PCDEx

PCDEx

30 HAL PWR Generic Driver

30.1 PWR Firmware driver registers structures

30.1.1 PWR_PVDTypeDef

PWR_PVDTypeDef is defined in the stm32f1xx_hal_pwr.h

Data Fields

- ***uint32_t PVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::PVDLevel*** PVDLevel: Specifies the PVD detection level. This parameter can be a value of ***PWR_PVD_detection_level***
- ***uint32_t PWR_PVDTypeDef::Mode*** Mode: Specifies the operating mode for the selected pins. This parameter can be a value of ***PWR_PVD_Mode***

30.2 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

30.2.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers) 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.
- ***HAL_PWR_DeInit()***
- ***HAL_PWR_EnableBkUpAccess()***
- ***HAL_PWR_DisableBkUpAccess()***

30.2.2 Peripheral Control functions

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the PWR_CR).
- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through __HAL_PVD_EXTI_ENABLE_IT() macro.
- The PVD is stopped in Standby mode.

WakeUp pin configuration

- WakeUp pin is used to wake up the system from Standby mode. This pin is forced in input pull-down configuration and is active on rising edges.
- There is one WakeUp pin: WakeUp Pin 1 on PA.00.

Low Power modes configuration

The device features 3 low-power modes:

- Sleep mode: CPU clock off, all peripherals including Cortex-M3 core peripherals like NVIC, SysTick, etc. are kept running
- Stop mode: All clocks are stopped
- Standby mode: 1.8V domain powered off

Sleep mode

- Entry: The Sleep mode is entered by using the HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFx) functions with
  - PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction
  - PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
- Exit:
  - WFI entry mode, Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.
  - WFE entry mode, Any wakeup event can wake up the device from Sleep mode.
    - Any peripheral interrupt w/o NVIC configuration & SEVONPEND bit set in the Cortex (HAL_PWR_EnableSEVOnPend)
    - Any EXTI Line (Internal or External) configured in Event mode

Stop mode

The Stop mode is based on the Cortex-M3 deepsleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the 1.8 V domain are stopped, the PLL, the HSI and the HSE RC oscillators are disabled. SRAM and register contents are preserved. In Stop mode, all I/O pins keep the same state as in Run mode.

- Entry: The Stop mode is entered using the HAL_PWR_EnterSTOPMode(PWR_REGULATOR_VALUE, PWR_SLEEPENTRY_WFx) function with:
  - PWR_REGULATOR_VALUE= PWR_MAINREGULATOR_ON: Main regulator ON.
  - PWR_REGULATOR_VALUE= PWR_LOWPOWERREGULATOR_ON: Low Power regulator ON.
  - PWR_SLEEPENTRY_WFx= PWR_SLEEPENTRY_WFI: enter STOP mode with WFI instruction
  - PWR_SLEEPENTRY_WFx= PWR_SLEEPENTRY_WFE: enter STOP mode with WFE instruction
- Exit:
  - WFI entry mode, Any EXTI Line (Internal or External) configured in Interrupt mode with NVIC configured

– WFE entry mode, Any EXTI Line (Internal or External) configured in Event mode.

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M3 deepsleep mode, with the voltage regulator disabled. The 1.8 V domain 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 registers in the Backup domain and Standby circuitry

- Entry:
  - The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function.
- Exit:
  - WKUP pin rising edge, RTC alarm event rising edge, external Reset in NRSTpin, IWDG Reset

Auto-wakeup (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm 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.

PWR Workarounds linked to Silicon Limitation

Below the list of all silicon limitations known on STM32F1xx prouct.

1. Workarounds Implemented inside PWR HAL Driver
   a. Debugging Stop mode with WFE entry - overloaded the WFE by an internal function

- ***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()***

30.2.3 HAL_PWR_DeInit

Function Name	**void HAL_PWR_DeInit (void)**
Function Description	Deinitializes the PWR peripheral registers to their default reset values.

Return values	• None

30.2.4 HAL_PWR_EnableBkUpAccess

Function Name	**void HAL_PWR_EnableBkUpAccess (void)**
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers).
Return values	• None
Notes	• If the HSE divided by 128 is used as the RTC clock, the Backup Domain Access should be kept enabled.

30.2.5 HAL_PWR_DisableBkUpAccess

Function Name	**void HAL_PWR_DisableBkUpAccess (void)**
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers).
Return values	• None
Notes	• If the HSE divided by 128 is used as the RTC clock, the Backup Domain Access should be kept enabled.

30.2.6 HAL_PWR_ConfigPVD

Function Name	**void HAL_PWR_ConfigPVD (PWR_PVDTypeDef * sConfigPVD)**
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	• **sConfigPVD:** pointer to an PWR_PVDTypeDef 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.

30.2.7 HAL_PWR_EnablePVD

Function Name	**void HAL_PWR_EnablePVD (void)**
Function Description	Enables the Power Voltage Detector(PVD).
Return values	• None

30.2.8 HAL_PWR_DisablePVD

Function Name	**void HAL_PWR_DisablePVD (void)**
Function Description	Disables the Power Voltage Detector(PVD).
Return values	• None

30.2.9 HAL_PWR_EnableWakeUpPin

Function Name	**void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)**
Function Description	Enables the WakeUp PINx functionality.
Parameters	• **WakeUpPinx:** Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values: PWR_WAKEUP_PIN1
Return values	• None

30.2.10 HAL_PWR_DisableWakeUpPin

Function Name	**void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)**
Function Description	Disables 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
Return values	• None

30.2.11 HAL_PWR_EnterSLEEPMode

Function Name	**void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)**
Function Description	Enters Sleep mode.
Parameters	• **Regulator:** Regulator state as no effect in SLEEP mode - allows to support portability from legacy software • **SLEEPEntry:** Specifies if SLEEP mode is entered with WFI or WFE instruction. When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source. 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	• In Sleep mode, all I/O pins keep the same state as in Run mode.

30.2.12 HAL_PWR_EnterSTOPMode

Function Name	**void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)**
Function Description	Enters 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 in entered with WFI or WFE instruction. This parameter can be one of the following values: PWR_STOPENTRY_WFI: Enter Stop mode with WFI instruction PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction
Return values	• None
Notes	• In Stop mode, all I/O pins keep the same state as in Run mode. • When exiting Stop mode by using an interrupt or a wakeup event, HSI RC oscillator is selected as system clock. • When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

30.2.13 HAL_PWR_EnterSTANDBYMode

Function Name	**void HAL_PWR_EnterSTANDBYMode (void)**
Function Description	Enters Standby mode.
Return values	• None
Notes	• In Standby mode, all I/O pins are high impedance except for: Reset pad (still available)TAMPER pin if configured for tamper or calibration out.WKUP pin (PA0) if enabled.

30.2.14 HAL_PWR_EnableSleepOnExit

Function Name	**void HAL_PWR_EnableSleepOnExit (void)**
Function Description	Indicates 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.

30.2.15 HAL_PWR_DisableSleepOnExit

Function Name	**void HAL_PWR_DisableSleepOnExit (void)**
Function Description	Disables 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.

30.2.16 HAL_PWR_EnableSEVOnPend

Function Name	**void HAL_PWR_EnableSEVOnPend (void)**
Function Description	Enables CORTEX M3 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.

30.2.17 HAL_PWR_DisableSEVOnPend

Function Name	**void HAL_PWR_DisableSEVOnPend (void)**
Function Description	Disables CORTEX M3 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.

30.2.18 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().

30.2.19 HAL_PWR_PVDCallback

Function Name	**void HAL_PWR_PVDCallback (void)**
Function Description	PWR PVD interrupt callback.

Return values • None

30.3 PWR Firmware driver defines

The following section lists the various define and macros of the module.

30.3.1 PWR

PWR

PWR CR Register alias address

LPSDSR_BIT_NUMBER

CR_LPSDSR_BB

DBP_BIT_NUMBER

CR_DBP_BB

PVDE_BIT_NUMBER

CR_PVDE_BB

PWR CSR Register alias address

CSR_EWUP_BB

PWR Exported Macros

__HAL_PWR_GET_FLAG

Description:

- Check PWR flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – PWR_FLAG_WU: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.
  – PWR_FLAG_SB: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
  – 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.

Return value:

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

__HAL_PWR_CLEAR_FLAG

Description:

- Clear the PWR's pending flags.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - PWR_FLAG_WU: Wake Up flag
  - PWR_FLAG_SB: StandBy flag

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:

- Enable interrupt on PVD Exti Line 16.

Return value:

- None.:

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:

- Disable interrupt on 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_FALLING_EDGE	**Description:** • PVD EXTI line configuration: set falling edge 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_EDGE	**Description:** • PVD EXTI line configuration: set rising edge 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_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:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None.:

PWR Flag

PWR_FLAG_WU

PWR_FLAG_SB

PWR_FLAG_PVDO

PWR Private Constants

PWR_EXTI_LINE_PVD	External interrupt line 16 Connected to the PVD EXTI Line

PWR Private Macros

IS_PWR_PVD_LEVEL

IS_PWR_PVD_MODE

IS_PWR_WAKEUP_PIN

IS_PWR_REGULATOR

IS_PWR_SLEEP_ENTRY

IS_PWR_STOP_ENTRY

PWR PVD detection level

PWR_PVDLEVEL_0

PWR_PVDLEVEL_1

PWR_PVDLEVEL_2

PWR_PVDLEVEL_3

PWR_PVDLEVEL_4

PWR_PVDLEVEL_5

PWR_PVDLEVEL_6

PWR_PVDLEVEL_7

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

PWR Register alias address

PWR_OFFSET

PWR_CR_OFFSET

PWR_CSR_OFFSET

PWR_CR_OFFSET_BB

PWR_CSR_OFFSET_BB

PWR Regulator state in SLEEP/STOP mode

PWR_MAINREGULATOR_ON

PWR_LOWPOWERREGULATOR_ON

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

PWR_SLEEPENTRY_WFE

PWR STOP mode entry

PWR_STOPENTRY_WFI

PWR_STOPENTRY_WFE

PWR WakeUp Pins

PWR_WAKEUP_PIN1

31 HAL RCC Generic Driver

31.1 RCC Firmware driver registers structures

31.1.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the stm32f1xx_hal_rcc.h

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLMUL***

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*** PLLSource: PLL entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLMUL*** PLLMUL: Multiplication factor for PLL VCO input clock This parameter must be a value of ***RCCEx_PLL_Multiplication_Factor***

31.1.2 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32f1xx_hal_rcc.h

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType*** The clock to be configured. This parameter can be a value of ***RCC_System_Clock_Type***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource*** The clock source (SYSCLKS) used as system clock. This parameter can be a value of ***RCC_System_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider*** The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_AHB_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider*** The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider*** The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_Clock_Source***

31.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

31.2.1 RCC specific features

After reset the device is running from Internal High Speed oscillator (HSI 8MHz) with Flash 0 wait state, Flash prefetch buffer is enabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses; all peripherals mapped on these busses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals whose clocks are not derived from the System clock (I2S, RTC, ADC, USB OTG FS)

31.2.2 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.
  - AHB & APB peripherals, 1 dummy read is necessary

Workarounds:

1. For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each __HAL_RCC_PPP_CLK_ENABLE() macro.

31.2.3 Initialization and de-initialization functions

This section provide functions allowing to configure the internal/external oscillators (HSE, HSI, LSE, LSI, PLL, CSS and MCO) and the System busses clocks (SYSCLK, AHB, APB1 and APB2).

Internal/external clock and PLL configuration

1. HSI (high-speed internal), 8 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. LSI (low-speed internal), 40 KHz low consumption RC used as IWDG and/or RTC clock source.
3. HSE (high-speed external), 4 to 24 MHz (STM32F100xx) or 4 to 16 MHz (STM32F101x/STM32F102x/STM32F103x) or 3 to 25 MHz (STM32F105x/STM32F107x) crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
4. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.

5. PLL (clocked by HSI or HSE), featuring two different output clocks:
   - The first output is used to generate the high speed system clock (up to 72 MHz for STM32F10xxx or up to 24 MHz for STM32F100xx)
   - The second output is used to generate the clock for the USB OTG FS (48 MHz)
6. CSS (Clock security system), once enable using the macro __HAL_RCC_CSS_ENABLE() and if a HSE clock failure occurs(HSE used directly or through PLL as System clock source), the System clockis automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.
7. MCO1 (microcontroller clock output), used to output SYSCLK, HSI, HSE or PLL clock (divided by 2) on PA8 pin + PLL2CLK, PLL3CLK/2, PLL3CLK and XTI for STM32F105x/STM32F107x

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "HAL_RCC_GetSysClockFreq()" function to retrieve the frequencies of these clocks. All the peripheral clocks are derived from the System clock (SYSCLK) except: RTC: RTC clock can be derived either from the LSI, LSE or HSE clock divided by 128. USB OTG FS and RTC: USB OTG FS require a frequency equal to 48 MHz to work correctly. This clock is derived of the main PLL through PLL Multiplier. I2S interface on STM32F105x/STM32F107x can be derived from PLL3CLK IWDG clock which is always the LSI clock.
2. For STM32F10xxx, the maximum frequency of the SYSCLK and HCLK/PCLK2 is 72 MHz, PCLK1 36 MHz. For STM32F100xx, the maximum frequency of the SYSCLK and HCLK/PCLK1/PCLK2 is 24 MHz. Depending on the SYSCLK frequency, the flash latency should be adapted (see *Table 18: "Number of wait states (WS) vs SYSCLK frequency"*).

Table 18: Number of wait states (WS) vs SYSCLK frequency

Latency	SYSCLK clock frequency (MHz)
0 WS (1 CPU cycle)	0 < SYSCLK ≤ 24
1 WS (2 CPU cycles)	24< SYSCLK ≤ 48
2 WS (3 CPU cycles)	48< SYSCLK ≤ 72

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

31.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

- ***HAL_RCC_MCOConfig()***
- ***HAL_RCC_EnableCSS()***
- ***HAL_RCC_DisableCSS()***
- ***HAL_RCC_GetSysClockFreq()***
- ***HAL_RCC_GetHCLKFreq()***

- *HAL_RCC_GetPCLK1Freq()*
- *HAL_RCC_GetPCLK2Freq()*
- *HAL_RCC_GetOscConfig()*
- *HAL_RCC_GetClockConfig()*
- *HAL_RCC_NMI_IRQHandler()*
- *HAL_RCC_CSSCallback()*

31.2.5 HAL_RCC_DeInit

Function Name	**void HAL_RCC_DeInit (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 and PLL OFFAHB, APB1 and APB2 prescaler set to 1.CSS and MCO1 OFFAll interrupts disabled • This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

31.2.6 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. • The PLL is not disabled when USB OTG FS clock is enabled (specific to devices with USB FS)

31.2.7 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 busses clocks according to the specified parameters in the RCC_ClkInitStruct.
Parameters	• **RCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral. • **FLatency:** FLASH Latency This parameter can be one of the following values: FLASH_LATENCY_0: FLASH 0 Latency cycle FLASH_LATENCY_1: FLASH 1 Latency cycle FLASH_LATENCY_2: FLASH 2 Latency cycle

	• **RCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral. • **FLatency:** FLASH Latency This parameter can be one of the following values: FLASH_LATENCY_0: FLASH 0 Latency cycle
Return values	• None • None
Notes	• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function • The HSI is used (enabled by hardware) as system clock source after startup 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 startup 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. • The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function • The HSI is used (enabled by hardware) as system clock source after startup 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 startup 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.

31.2.8 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 MCO pin.
Parameters	• **RCC_MCOx:** specifies the output direction for the clock source. This parameter can be one of the following values: RCC_MCO: Clock source to output on MCO1 pin(PA8). • **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values: RCC_MCO1SOURCE_NOCLOCK: No clock selected RCC_MCO1SOURCE_SYSCLK: System clock selected as MCO source RCC_MCO1SOURCE_HSI: HSI oscillator clock selected RCC_MCO1SOURCE_HSE: HSE oscillator clock

selected RCC_MCO1SOURCE_PLLCLK: PLL clock divided by 2 selected as MCO source RCC_MCO1SOURCE_PLL2CLK: PLL2 clock selected as MCO source (only for connectivity line devices) RCC_MCO1SOURCE_PLL3CLK_DIV2: PLL3 clock divided by 2 selected as MCO source (only for connectivity line devices) RCC_MCO1SOURCE_EXT_HSE: XT1 external 3-25 MHz oscillator clock selected as MCO source (only for connectivity line devices) RCC_MCO1SOURCE_PLL3CLK: PLL3 clock selected as MCO source (only for connectivity line devices)

- **RCC_MCODiv:** specifies the MCO DIV. This parameter can be one of the following values: RCC_MCODIV_1: no division applied to MCO clock

Return values
- None

Notes
- MCO pin should be configured in alternate function mode.

31.2.9 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-M3 NMI (Non-Maskable Interrupt) exception vector.

31.2.10 HAL_RCC_DisableCSS

Function Name	**void HAL_RCC_DisableCSS (void)**
Function Description	Disables the Clock Security System.
Return values	• None

31.2.11 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 HSI, function returns values based on HSI_VALUE(*)

- If SYSCLK source is HSE, function returns values based on HSE_VALUE divided by PREDIV factor(**)
- If SYSCLK source is PLL, function returns values based on HSE_VALUE divided by PREDIV factor(**) or HSI_VALUE(*) multiplied by the PLL factor.
- (*) HSI_VALUE is a constant defined in stm32f1xx_hal_conf.h file (default value 8 MHz).
- (**) HSE_VALUE is a constant defined in stm32f1xx_hal_conf.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

31.2.12 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 SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

31.2.13 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.

31.2.14 HAL_RCC_GetPCLK2Freq

Function Name	**uint32_t HAL_RCC_GetPCLK2Freq (void)**
Function Description	Returns the PCLK2 frequency.
Return values	• PCLK2 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.

31.2.15 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

31.2.16 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

31.2.17 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().

31.2.18 HAL_RCC_CSSCallback

Function Name	**void HAL_RCC_CSSCallback (void)**
Function Description	RCC Clock Security System interrupt callback.
Return values	• none

31.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

31.3.1 RCC

RCC

AHB Clock Source

RCC_SYSCLK_DIV1	SYSCLK not divided
RCC_SYSCLK_DIV2	SYSCLK divided by 2
RCC_SYSCLK_DIV4	SYSCLK divided by 4
RCC_SYSCLK_DIV8	SYSCLK divided by 8
RCC_SYSCLK_DIV16	SYSCLK divided by 16
RCC_SYSCLK_DIV64	SYSCLK divided by 64
RCC_SYSCLK_DIV128	SYSCLK divided by 128
RCC_SYSCLK_DIV256	SYSCLK divided by 256
RCC_SYSCLK_DIV512	SYSCLK divided by 512

AHB Peripheral Clock Enable Disable Status

__HAL_RCC_DMA1_IS_CLK_ENABLED

__HAL_RCC_DMA1_IS_CLK_DISABLED

__HAL_RCC_SRAM_IS_CLK_ENABLED

__HAL_RCC_SRAM_IS_CLK_DISABLED

__HAL_RCC_FLITF_IS_CLK_ENABLED

__HAL_RCC_FLITF_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

Alias define maintained for legacy

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

APB1 APB2 Clock Source

RCC_HCLK_DIV1	HCLK not divided
RCC_HCLK_DIV2	HCLK divided by 2
RCC_HCLK_DIV4	HCLK divided by 4
RCC_HCLK_DIV8	HCLK divided by 8
RCC_HCLK_DIV16	HCLK divided by 16

APB1 Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_WWDG_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_BKP_CLK_ENABLE

__HAL_RCC_PWR_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_WWDG_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_BKP_CLK_DISABLE

__HAL_RCC_PWR_CLK_DISABLE

APB1 Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_WWDG_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_BKP_FORCE_RESET

__HAL_RCC_PWR_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_WWDG_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_BKP_RELEASE_RESET

__HAL_RCC_PWR_RELEASE_RESET

APB1 Peripheral Clock Enable Disable Status

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_WWDG_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_BKP_IS_CLK_ENABLED

__HAL_RCC_BKP_IS_CLK_DISABLED

__HAL_RCC_PWR_IS_CLK_ENABLED

__HAL_RCC_PWR_IS_CLK_DISABLED

APB2 Clock Enable Disable

__HAL_RCC_AFIO_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_ADC1_CLK_ENABLE

__HAL_RCC_TIM1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_AFIO_CLK_DISABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_ADC1_CLK_DISABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

APB2 Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_AFIO_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_ADC1_FORCE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_AFIO_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_ADC1_RELEASE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

APB2 Peripheral Clock Enable Disable Status

__HAL_RCC_AFIO_IS_CLK_ENABLED

__HAL_RCC_AFIO_IS_CLK_DISABLED

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_DISABLED

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_DISABLED

__HAL_RCC_ADC1_IS_CLK_ENABLED

__HAL_RCC_ADC1_IS_CLK_DISABLED

__HAL_RCC_TIM1_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

BitAddress AliasRegion

RCC_OFFSET

RCC_CR_OFFSET

RCC_CFGR_OFFSET

RCC_CIR_OFFSET

RCC_BDCR_OFFSET

RCC_CSR_OFFSET

RCC_CR_OFFSET_BB

RCC_CFGR_OFFSET_BB

RCC_CIR_OFFSET_BB

RCC_BDCR_OFFSET_BB

RCC_CSR_OFFSET_BB

HSION_BITNUMBER

RCC_CR_HSION_BB

HSEON_BITNUMBER

CR_HSEON_BB

CSSON_BITNUMBER

RCC_CR_CSSON_BB

PLLON_BITNUMBER

RCC_CR_PLLON_BB

LSION_BITNUMBER

RCC_CSR_LSION_BB

LSEON_BITNUMBER

BDCR_LSEON_BB

LSEBYP_BITNUMBER

BDCR_LSEBYP_BB

RTCEN_BITNUMBER

RCC_BDCR_RTCEN_BB

BDRST_BITNUMBER

RCC_BDCR_BDRST_BB

RCC_CR_BYTE2_ADDRESS

RCC_CIR_BYTE1_ADDRESS

RCC_CIR_BYTE2_ADDRESS

CR_REG_INDEX

BDCR_REG_INDEX

CSR_REG_INDEX

RCC_FLAG_MASK

Flags

RCC_FLAG_HSIRDY	Internal High Speed clock ready flag
RCC_FLAG_HSERDY	External High Speed clock ready flag
RCC_FLAG_PLLRDY	PLL clock ready flag
RCC_FLAG_LSERDY	External Low Speed oscillator Ready

RCC_FLAG_LSIRDY	Internal Low Speed oscillator Ready
RCC_FLAG_RMV	Remove reset flag
RCC_FLAG_PINRST	PIN reset flag
RCC_FLAG_PORRST	POR/PDR reset flag
RCC_FLAG_SFTRST	Software Reset flag
RCC_FLAG_IWDGRST	Independent Watchdog reset flag
RCC_FLAG_WWDGRST	Window watchdog reset flag
RCC_FLAG_LPWRRST	Low-Power reset flag

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to enable the selected interrupts.).

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY: LSI ready interrupt
  - RCC_IT_LSERDY: LSE ready interrupt
  - RCC_IT_HSIRDY: HSI ready interrupt
  - RCC_IT_HSERDY: HSE ready interrupt
  - RCC_IT_PLLRDY: main PLL ready interrupt
  - RCC_IT_PLL2RDY: Main PLL2 ready interrupt.(*)
  - RCC_IT_PLLI2S2RDY: Main PLLI2S ready interrupt.(*)

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to disable the selected interrupts).

Parameters:

- __INTERRUPT__: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY: LSI ready interrupt
  - RCC_IT_LSERDY: LSE ready interrupt
  - RCC_IT_HSIRDY: HSI ready interrupt
  - RCC_IT_HSERDY: HSE ready interrupt
  - RCC_IT_PLLRDY: main PLL ready interrupt
  - RCC_IT_PLL2RDY: Main PLL2 ready

interrupt.(*)
- RCC_IT_PLLI2S2RDY: Main PLLI2S ready interrupt.(*)

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC_CIR[23:16] bits to clear the selected interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY: LSI ready interrupt.
  - RCC_IT_LSERDY: LSE ready interrupt.
  - RCC_IT_HSIRDY: HSI ready interrupt.
  - RCC_IT_HSERDY: HSE ready interrupt.
  - RCC_IT_PLLRDY: Main PLL ready interrupt.
  - RCC_IT_PLL2RDY: Main PLL2 ready interrupt.(*)
  - RCC_IT_PLLI2S2RDY: Main PLLI2S ready 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 one of the following values:
  - RCC_IT_LSIRDY: LSI ready interrupt.
  - RCC_IT_LSERDY: LSE ready interrupt.
  - RCC_IT_HSIRDY: HSI ready interrupt.
  - RCC_IT_HSERDY: HSE ready interrupt.
  - RCC_IT_PLLRDY: Main PLL ready interrupt.
  - RCC_IT_PLL2RDY: Main PLL2 ready interrupt.(*)
  - RCC_IT_PLLI2S2RDY: Main PLLI2S ready interrupt.(*)
  - RCC_IT_CSS: Clock Security System interrupt

Return value:

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

__HAL_RCC_CLEAR_RESET_FLAGS

__HAL_RCC_GET_FLAG

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_HSERDY: HSE oscillator clock ready.
  - RCC_FLAG_PLLRDY: Main PLL clock ready.
  - RCC_FLAG_PLL2RDY: Main PLL2 clock ready.(*)
  - RCC_FLAG_PLLI2SRDY: Main PLLI2S clock ready.(*)
  - RCC_FLAG_LSERDY: LSE oscillator clock ready.
  - RCC_FLAG_LSIRDY: LSI oscillator clock ready.
  - RCC_FLAG_PINRST: Pin reset.
  - RCC_FLAG_PORRST: POR/PDR reset.
  - RCC_FLAG_SFTRST: Software reset.
  - RCC_FLAG_IWDGRST: Independent Watchdog reset.
  - RCC_FLAG_WWDGRST: Window Watchdog reset.
  - RCC_FLAG_LPWRRST: Low Power reset.

Return value:

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

Get Clock source

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
  - RCC_SYSCLKSOURCE_STATUS_HSI: HSI used as system clock
  - RCC_SYSCLKSOURCE_STATUS_HSE: HSE used as system clock
  - RCC_SYSCLKSOURCE_STATUS_PLLCLK: PLL used as system clock

__HAL_RCC_GET_PLL_OSCSOURCE

Description:

- Get oscillator clock selected as PLL input clock.

Return value:

- The: clock source used for PLL entry. The returned value can be one of the following:
  - RCC_PLLSOURCE_HSI_DIV2: HSI oscillator clock selected as PLL input clock
  - RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL input clock

HSE Config

RCC_HSE_OFF	HSE clock deactivation
RCC_HSE_ON	HSE clock activation
RCC_HSE_BYPASS	External clock source for HSE clock

HSE Configuration

__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

HSI Config

RCC_HSI_OFF	HSI clock deactivation
RCC_HSI_ON	HSI clock activation
RCC_HSICALIBRATION_DEFAULT	

HSI Configuration

__HAL_RCC_HSI_ENABLE

__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. (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 0x1F.

Interrupts

RCC_IT_LSIRDY	LSI Ready Interrupt flag
RCC_IT_LSERDY	LSE Ready Interrupt flag
RCC_IT_HSIRDY	HSI Ready Interrupt flag
RCC_IT_HSERDY	HSE Ready Interrupt flag
RCC_IT_PLLRDY	PLL Ready Interrupt flag
RCC_IT_CSS	Clock Security System Interrupt flag

LSE Config

RCC_LSE_OFF	LSE clock deactivation
RCC_LSE_ON	LSE clock activation
RCC_LSE_BYPASS	External clock source for LSE clock

LSE Configuration

__HAL_RCC_LSE_CONFIG

LSI Config

RCC_LSI_OFF	LSI clock deactivation
RCC_LSI_ON	LSI clock activation

LSI Configuration

__HAL_RCC_LSI_ENABLE

__HAL_RCC_LSI_DISABLE

MCO1 Clock Prescaler

RCC_MCODIV_1

MCO Index

RCC_MCO1	
RCC_MCO	MCO1 to be compliant with other families with 2 MCOs

Oscillator Type

RCC_OSCILLATORTYPE_NONE

RCC_OSCILLATORTYPE_HSE

RCC_OSCILLATORTYPE_HSI

RCC_OSCILLATORTYPE_LSE

RCC_OSCILLATORTYPE_LSI

Peripheral Clock Enable Disable

__HAL_RCC_DMA1_CLK_ENABLE

__HAL_RCC_SRAM_CLK_ENABLE

__HAL_RCC_FLITF_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_DMA1_CLK_DISABLE

__HAL_RCC_SRAM_CLK_DISABLE

__HAL_RCC_FLITF_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

PLL Clock Source

RCC_PLLSOURCE_HSI_DIV2	HSI clock divided by 2 selected as PLL entry clock source
RCC_PLLSOURCE_HSE	HSE clock selected as PLL entry clock source

PLL Config

RCC_PLL_NONE	PLL is not configured
RCC_PLL_OFF	PLL deactivation
RCC_PLL_ON	PLL activation

PLL Configuration

__HAL_RCC_PLL_ENABLE

__HAL_RCC_PLL_DISABLE

__HAL_RCC_PLL_CONFIG

Description:

- macros to configure the main PLL clock source and multiplication factors.

Parameters:

- __RCC_PLLSOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_HSI_DIV2: HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL clock entry
- __PLLMUL__: specifies the multiplication factor for PLL VCO output clock This parameter can be one of the following values:
  - RCC_PLL_MUL2: PLLVCO = PLL clock entry x 2 (*)
  - RCC_PLL_MUL3: PLLVCO = PLL clock entry x 3 (*)
  - RCC_PLL_MUL4: PLLVCO = PLL clock entry x 4
  - RCC_PLL_MUL6: PLLVCO = PLL clock entry x 6
  - RCC_PLL_MUL6_5: PLLVCO = PLL clock entry x 6.5 (**)
  - RCC_PLL_MUL8: PLLVCO = PLL clock entry x 8
  - RCC_PLL_MUL9: PLLVCO = PLL clock entry x 9
  - RCC_PLL_MUL10: PLLVCO = PLL clock entry x 10 (*)
  - RCC_PLL_MUL11: PLLVCO = PLL clock entry x 11 (*)
  - RCC_PLL_MUL12: PLLVCO = PLL clock entry x 12 (*)
  - RCC_PLL_MUL13: PLLVCO = PLL clock entry x 13 (*)
  - RCC_PLL_MUL14: PLLVCO = PLL clock entry x 14 (*)

– RCC_PLL_MUL15: PLLVCO = PLL clock entry x 15 (*)
– RCC_PLL_MUL16: PLLVCO = PLL clock entry x 16 (*)

RCC Private Constants

RCC_DBP_TIMEOUT_VALUE

RCC_LSE_TIMEOUT_VALUE

CLOCKSWITCH_TIMEOUT_VALUE

HSE_TIMEOUT_VALUE

HSI_TIMEOUT_VALUE

LSI_TIMEOUT_VALUE

PLL_TIMEOUT_VALUE

LSI_VALUE

RCC Private Macros

MCO1_CLK_ENABLE

MCO1_GPIO_PORT

MCO1_PIN

IS_RCC_HSI

IS_RCC_CALIBRATION_VALUE

IS_RCC_CLOCKTYPE

IS_RCC_HSE

IS_RCC_LSE

IS_RCC_PLLSOURCE

IS_RCC_OSCILLATORTYPE

IS_RCC_LSI

IS_RCC_PLL

IS_RCC_SYSCLKSOURCE

IS_RCC_HCLK

IS_RCC_PCLK

IS_RCC_MCO

IS_RCC_MCODIV

RCC RTC Clock Configuration

__HAL_RCC_RTC_CONFIG

Description:

- Macro to configures the RTC clock (RTCCLK).

Parameters:

- __RTC_CLKSOURCE__: 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_DIV128 : HSE divided by 128 selected as RTC clock

__HAL_RCC_GET_RTC_SOURCE

__HAL_RCC_RTC_ENABLE

__HAL_RCC_RTC_DISABLE

__HAL_RCC_BACKUPRESET_FORCE

__HAL_RCC_BACKUPRESET_RELEASE

RTC Clock Source

RCC_RTCCLKSOURCE_LSE	LSE oscillator clock used as RTC clock
RCC_RTCCLKSOURCE_LSI	LSI oscillator clock used as RTC clock
RCC_RTCCLKSOURCE_HSE_DIV128	HSE oscillator clock divided by 128 used as RTC clock

System Clock Source

RCC_SYSCLKSOURCE_HSI	HSI selected as system clock
RCC_SYSCLKSOURCE_HSE	HSE selected as system clock
RCC_SYSCLKSOURCE_PLLCLK	PLL selected as system clock

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_HSI

RCC_SYSCLKSOURCE_STATUS_HSE

RCC_SYSCLKSOURCE_STATUS_PLLCLK

System Clock Type

RCC_CLOCKTYPE_SYSCLK	SYSCLK to configure
RCC_CLOCKTYPE_HCLK	HCLK to configure
RCC_CLOCKTYPE_PCLK1	PCLK1 to configure
RCC_CLOCKTYPE_PCLK2	PCLK2 to configure

32 HAL RCC Extension Driver

32.1 RCCEx Firmware driver registers structures

32.1.1 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32f1xx_hal_rcc_ex.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t HSEPredivValue***
- ***uint32_t LSEState***
- ***uint32_t HSIState***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***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::HSEPredivValue*** The Prediv1 factor value (named PREDIV1 or PLLXTPRE in RM) This parameter can be a value of ***RCCEx_Prediv1_Factor***
- ***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 HSI calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F
- ***uint32_t RCC_OscInitTypeDef::LSIState*** The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL*** PLL structure parameters

32.1.2 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32f1xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PeriphClockSelection***
- ***uint32_t RTCClockSelection***
- ***uint32_t AdcClockSelection***
- ***uint32_t I2s2ClockSelection***
- ***uint32_t I2s3ClockSelection***
- ***uint32_t UsbClockSelection***

Field Documentation

- ***uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection*** The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection*** specifies the RTC clock source. This parameter can be a value of ***RCC_RTC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcClockSelection*** ADC clock source This parameter can be a value of ***RCCEx_ADC_Prescaler***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2s2ClockSelection*** I2S2 clock source This parameter can be a value of ***RCCEx_I2S2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2s3ClockSelection*** I2S3 clock source This parameter can be a value of ***RCCEx_I2S3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbClockSelection*** USB clock source This parameter can be a value of ***RCCEx_USB_Prescaler***

32.2 RCCEx Firmware driver API description

The following section lists the various functions of the RCCEx library.

32.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.

- *HAL_RCCEx_PeriphCLKConfig()*
- *HAL_RCCEx_GetPeriphCLKConfig()*
- *HAL_RCCEx_GetPeriphCLKFreq()*

32.2.2 HAL_RCCEx_PeriphCLKConfig

Function Name	**HAL_StatusTypeDef HAL_RCCEx_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(RTC clock).
Return values	• HAL status
Notes	• Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the 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.
- In case of STM32F105xC or STM32F107xC devices, PLLI2S will be enabled if requested on one of 2 I2S interfaces. When PLLI2S is enabled, you need to call HAL_RCCEx_DisablePLLI2S to manually disable it.

32.2.3 HAL_RCCEx_GetPeriphCLKConfig

Function Name	**void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)**
Function Description	Get the PeriphClkInit 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(RTC, I2S, ADC clocks).
Return values	• None

32.2.4 HAL_RCCEx_GetPeriphCLKFreq

Function Name	**uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_t PeriphClk)**
Function Description	Returns the peripheral clock frequency.
Parameters	• **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values: RCC_PERIPHCLK_RTC: RTC peripheral clock RCC_PERIPHCLK_ADC: ADC peripheral clock RCC_PERIPHCLK_I2S2: I2S2 peripheral clock (STM32F103xE, STM32F103xG, STM32F105xC & STM32F107xC) RCC_PERIPHCLK_I2S3: I2S3 peripheral clock (STM32F103xE, STM32F103xG, STM32F105xC & STM32F107xC) RCC_PERIPHCLK_USB: USB peripheral clock (STM32F102xx, STM32F103xx, STM32F105xC & STM32F107xC)
Return values	• Frequency in Hz (0: means that no available frequency for the peripheral)
Notes	• Returns 0 if peripheral clock is unknown

32.3 RCCEx Firmware driver defines

The following section lists the various define and macros of the module.

32.3.1 RCCEx

RCCEx

ADC Prescaler

RCC_ADCPCLK2_DIV2

RCC_ADCPCLK2_DIV4

RCC_ADCPCLK2_DIV6

RCC_ADCPCLK2_DIV8

AHB1 Peripheral Clock Enable Disable Status

__HAL_RCC_DMA2_IS_CLK_ENABLED

__HAL_RCC_DMA2_IS_CLK_DISABLED

__HAL_RCC_FSMC_IS_CLK_ENABLED

__HAL_RCC_FSMC_IS_CLK_DISABLED

__HAL_RCC_SDIO_IS_CLK_ENABLED

__HAL_RCC_SDIO_IS_CLK_DISABLED

APB1 Clock Enable Disable

__HAL_RCC_CAN1_CLK_ENABLE

__HAL_RCC_CAN1_CLK_DISABLE

__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_TIM4_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_USB_CLK_ENABLE

__HAL_RCC_USB_CLK_DISABLE

__HAL_RCC_TIM5_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_UART4_CLK_ENABLE

__HAL_RCC_UART5_CLK_ENABLE

__HAL_RCC_DAC_CLK_ENABLE

__HAL_RCC_TIM5_CLK_DISABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_UART4_CLK_DISABLE

__HAL_RCC_UART5_CLK_DISABLE

__HAL_RCC_DAC_CLK_DISABLE

__HAL_RCC_TIM12_CLK_ENABLE

__HAL_RCC_TIM13_CLK_ENABLE

__HAL_RCC_TIM14_CLK_ENABLE

__HAL_RCC_TIM12_CLK_DISABLE

__HAL_RCC_TIM13_CLK_DISABLE

__HAL_RCC_TIM14_CLK_DISABLE

APB1 Force Release Reset

__HAL_RCC_CAN1_FORCE_RESET

__HAL_RCC_CAN1_RELEASE_RESET

__HAL_RCC_TIM4_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_TIM4_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_USB_FORCE_RESET

__HAL_RCC_USB_RELEASE_RESET

__HAL_RCC_TIM5_FORCE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_UART4_FORCE_RESET

__HAL_RCC_UART5_FORCE_RESET

__HAL_RCC_DAC_FORCE_RESET

__HAL_RCC_TIM5_RELEASE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_UART4_RELEASE_RESET

__HAL_RCC_UART5_RELEASE_RESET

__HAL_RCC_DAC_RELEASE_RESET

__HAL_RCC_TIM12_FORCE_RESET

__HAL_RCC_TIM13_FORCE_RESET

__HAL_RCC_TIM14_FORCE_RESET

__HAL_RCC_TIM12_RELEASE_RESET

__HAL_RCC_TIM13_RELEASE_RESET

__HAL_RCC_TIM14_RELEASE_RESET

APB1 Peripheral Clock Enable Disable Status

__HAL_RCC_CAN1_IS_CLK_ENABLED

__HAL_RCC_CAN1_IS_CLK_DISABLED

__HAL_RCC_TIM4_IS_CLK_ENABLED

__HAL_RCC_TIM4_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_USB_IS_CLK_ENABLED

__HAL_RCC_USB_IS_CLK_DISABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED

__HAL_RCC_TIM5_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_DISABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_DISABLED

__HAL_RCC_UART4_IS_CLK_ENABLED

__HAL_RCC_UART4_IS_CLK_DISABLED

__HAL_RCC_UART5_IS_CLK_ENABLED

__HAL_RCC_UART5_IS_CLK_DISABLED

__HAL_RCC_DAC_IS_CLK_ENABLED

__HAL_RCC_DAC_IS_CLK_DISABLED

__HAL_RCC_TIM13_IS_CLK_ENABLED

__HAL_RCC_TIM13_IS_CLK_DISABLED

__HAL_RCC_TIM14_IS_CLK_ENABLED

__HAL_RCC_TIM14_IS_CLK_DISABLED

APB2 Clock Enable Disable

__HAL_RCC_ADC2_CLK_ENABLE

__HAL_RCC_ADC2_CLK_DISABLE

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOG_CLK_ENABLE

__HAL_RCC_GPIOF_CLK_DISABLE

__HAL_RCC_GPIOG_CLK_DISABLE

__HAL_RCC_TIM8_CLK_ENABLE

__HAL_RCC_ADC3_CLK_ENABLE

__HAL_RCC_TIM8_CLK_DISABLE

__HAL_RCC_ADC3_CLK_DISABLE

__HAL_RCC_TIM9_CLK_ENABLE

__HAL_RCC_TIM10_CLK_ENABLE

__HAL_RCC_TIM11_CLK_ENABLE

__HAL_RCC_TIM9_CLK_DISABLE

__HAL_RCC_TIM10_CLK_DISABLE

__HAL_RCC_TIM11_CLK_DISABLE

APB2 Force Release Reset

__HAL_RCC_ADC2_FORCE_RESET

__HAL_RCC_ADC2_RELEASE_RESET

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_GPIOG_FORCE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

__HAL_RCC_GPIOG_RELEASE_RESET

__HAL_RCC_TIM8_FORCE_RESET

__HAL_RCC_ADC3_FORCE_RESET

__HAL_RCC_TIM8_RELEASE_RESET

__HAL_RCC_ADC3_RELEASE_RESET

__HAL_RCC_TIM9_FORCE_RESET

__HAL_RCC_TIM10_FORCE_RESET

__HAL_RCC_TIM11_FORCE_RESET

__HAL_RCC_TIM9_RELEASE_RESET

__HAL_RCC_TIM10_RELEASE_RESET

__HAL_RCC_TIM11_RELEASE_RESET

APB2 Peripheral Clock Enable Disable Status

__HAL_RCC_ADC2_IS_CLK_ENABLED

__HAL_RCC_ADC2_IS_CLK_DISABLED

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_DISABLED

__HAL_RCC_GPIOF_IS_CLK_ENABLED

__HAL_RCC_GPIOF_IS_CLK_DISABLED

__HAL_RCC_GPIOG_IS_CLK_ENABLED

__HAL_RCC_GPIOG_IS_CLK_DISABLED

__HAL_RCC_TIM8_IS_CLK_ENABLED

__HAL_RCC_TIM8_IS_CLK_DISABLED

__HAL_RCC_ADC3_IS_CLK_ENABLED

__HAL_RCC_ADC3_IS_CLK_DISABLED

__HAL_RCC_TIM9_IS_CLK_ENABLED

__HAL_RCC_TIM9_IS_CLK_DISABLED

__HAL_RCC_TIM10_IS_CLK_ENABLED

__HAL_RCC_TIM10_IS_CLK_DISABLED

__HAL_RCC_TIM11_IS_CLK_ENABLED

__HAL_RCC_TIM11_IS_CLK_DISABLED

HSE Configuration

__HAL_RCC_HSE_PREDIV_CONFIG

Description:

- Macro to configure the External High Speed oscillator (HSE) Predivision factor for PLL.

Parameters:

- __HSE_PREDIV_VALUE__: specifies the division value applied to HSE. This parameter must be a number between RCC_HSE_PREDIV_DIV1 and RCC_HSE_PREDIV_DIV2.

__HAL_RCC_HSE_GET_PREDIV

I2S2 Clock Source

RCC_I2S2CLKSOURCE_SYSCLK

I2S3 Clock Source

RCC_I2S3CLKSOURCE_SYSCLK

MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK

RCC_MCO1SOURCE_SYSCLK

RCC_MCO1SOURCE_HSI

RCC_MCO1SOURCE_HSE

RCC_MCO1SOURCE_PLLCLK

Peripheral Clock Enable Disable

__HAL_RCC_DMA2_CLK_ENABLE

__HAL_RCC_DMA2_CLK_DISABLE

__HAL_RCC_FSMC_CLK_ENABLE

__HAL_RCC_FSMC_CLK_DISABLE

__HAL_RCC_SDIO_CLK_ENABLE

__HAL_RCC_SDIO_CLK_DISABLE

Peripheral Configuration

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock.

Parameters:

- __USBCLKSOURCE__: specifies the USB clock source. This parameter can be one of the following values:
  - RCC_USBPLLCLK_DIV1: PLL clock divided by 1 selected as USB clock
  - RCC_USBPLLCLK_DIV1_5: PLL clock divided by 1.5 selected as USB clock

__HAL_RCC_GET_USB_SOURCE

Description:

- Macro to get the USB clock (USBCLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_USBPLLCLK_DIV1: PLL clock divided by 1 selected as USB clock
  - RCC_USBPLLCLK_DIV1_5: PLL clock divided by 1.5 selected as USB clock

__HAL_RCC_ADC_CONFIG

Description:

- Macro to configure the ADCx clock (x=1 to 3 depending on devices).

Parameters:

- __ADCCLKSOURCE__: specifies the ADC clock source. This parameter can be one of the following values:
  - RCC_ADCPCLK2_DIV2: PCLK2 clock divided by 2 selected as ADC clock
  - RCC_ADCPCLK2_DIV4: PCLK2 clock divided by 4 selected as ADC clock
  - RCC_ADCPCLK2_DIV6: PCLK2 clock divided by 6 selected as ADC clock
  - RCC_ADCPCLK2_DIV8: PCLK2 clock divided by 8 selected as ADC clock

__HAL_RCC_GET_ADC_SOURCE	**Description:** • Macro to get the ADC clock (ADCxCLK, x=1 to 3 depending on devices). **Return value:** • The: clock source can be one of the following values: – RCC_ADCPCLK2_DIV2: PCLK2 clock divided by 2 selected as ADC clock – RCC_ADCPCLK2_DIV4: PCLK2 clock divided by 4 selected as ADC clock – RCC_ADCPCLK2_DIV6: PCLK2 clock divided by 6 selected as ADC clock – RCC_ADCPCLK2_DIV8: PCLK2 clock divided by 8 selected as ADC clock

Periph Clock Selection

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_ADC

RCC_PERIPHCLK_I2S2

RCC_PERIPHCLK_I2S3

RCC_PERIPHCLK_USB

PLL Multiplication Factor

RCC_PLL_MUL2

RCC_PLL_MUL3

RCC_PLL_MUL4

RCC_PLL_MUL5

RCC_PLL_MUL6

RCC_PLL_MUL7

RCC_PLL_MUL8

RCC_PLL_MUL9

RCC_PLL_MUL10

RCC_PLL_MUL11

RCC_PLL_MUL12

RCC_PLL_MUL13

RCC_PLL_MUL14

RCC_PLL_MUL15

RCC_PLL_MUL16

HSE Prediv1 Factor

RCC_HSE_PREDIV_DIV1

RCC_HSE_PREDIV_DIV2

RCCEx Private Constants

PLL2_TIMEOUT_VALUE

PLL2ON_BITNUMBER

CR_PLL2ON_BB

CR_REG_INDEX

RCCEx Private Macros

IS_RCC_HSE_PREDIV

IS_RCC_PLL_MUL

IS_RCC_MCO1SOURCE

IS_RCC_ADCPLLCLK_DIV

IS_RCC_I2S2CLKSOURCE

IS_RCC_I2S3CLKSOURCE

IS_RCC_PERIPHCLOCK

IS_RCC_USBPLLCLK_DIV

USB Prescaler

RCC_USBPLLCLK_DIV1

RCC_USBPLLCLK_DIV1_5

33 HAL RTC Generic Driver

33.1 RTC Firmware driver registers structures

33.1.1 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32f1xx_hal_rtc.h

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

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 = 23
- ***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

33.1.2 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32f1xx_hal_rtc.h

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime*** Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::Alarm*** Specifies the alarm ID (only 1 alarm ID for STM32F1). This parameter can be a value of ***RTC_Alarms_Definitions***

33.1.3 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32f1xx_hal_rtc.h

Data Fields

- ***uint32_t AsynchPrediv***
- ***uint32_t OutPut***

Field Documentation

- ***uint32_t RTC_InitTypeDef::AsynchPrediv*** Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFFF or RTC_AUTO_1_SECOND If RTC_AUTO_1_SECOND is selected, AsynchPrediv will be set automatically to get 1sec timebase
- ***uint32_t RTC_InitTypeDef::OutPut*** Specifies which signal will be routed to the RTC Tamper pin. This parameter can be a value of ***RTC_output_source_to_output_on_the_Tamper_pin***

33.1.4 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32f1xx_hal_rtc.h

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 (not necessary for HAL_RTC_SetDate). 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

33.1.5 RTC_HandleTypeDef

RTC_HandleTypeDef is defined in the stm32f1xx_hal_rtc.h

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***RTC_DateTypeDef DateToUpdate***
- ***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
- ***RTC_DateTypeDef RTC_HandleTypeDef::DateToUpdate*** Current date set by user and updated automatically
- ***HAL_LockTypeDef RTC_HandleTypeDef::Lock*** RTC locking object
- ***__IO HAL_RTCStateTypeDef RTC_HandleTypeDef::State*** Time communication state

33.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

33.2.1 How to use this driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous prescaler to generate RTC 1Hz time base) using the HAL_RTC_Init() function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

- 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.
- To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper Level using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The TAMPER1 alternate function can be mapped to PC13

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.

33.2.2 WARNING: Drivers Restrictions

RTC version used on STM32F1 families is version V1. All the features supported by V2 (other families) will be not supported on F1.

As on V2, main RTC features are managed by HW. But on F1, date feature is completely managed by SW.

Then, there are some restrictions compared to other families:

- Only format 24 hours supported in HAL (format 12 hours not supported)
- Date is saved in SRAM. Then, when MCU is in STOP or STANDBY mode, date will be lost. User should implement a way to save date before entering in low power mode (an example is provided with firmware package based on backup registers)
- Date is automatically updated each time a HAL_RTC_GetTime or HAL_RTC_GetDate is called.
- Alarm detection is limited to 1 day. It will expire only 1 time (no alarm repetition, need to program a new alarm)

33.2.3 Backup Domain 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.

To allow the RTC operating even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

- The RTC
- The LSE oscillator
- PC13 I/O

When the backup domain is supplied by VDD (analog switch connected to VDD), the following pins are available:

- PC13 can be used as a Tamper pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following pins are available:

- PC13 can be used as the Tamper pin

33.2.4 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.

33.2.5 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:

- Call the function HAL_RCCEx_PeriphCLKConfig in using RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSE)
- Enable the BKP clock in using __HAL_RCC_BKP_CLK_ENABLE()

33.2.6 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 RTC tamper 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.

33.2.7 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Asynchronous), 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 should be programmed to generate the RTC 1Hz time base.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by setting the CNF bit in the RTC_CRL register.
3. To read the calendar after wakeup from low power modes (Standby or Stop) the software must first wait for the RSF bit (Register Synchronized Flag) in the RTC_CRL register to be set by hardware. The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

- ***HAL_RTC_Init()***
- ***HAL_RTC_DeInit()***
- ***HAL_RTC_MspInit()***
- ***HAL_RTC_MspDeInit()***

33.2.8 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

- ***HAL_RTC_SetTime()***
- ***HAL_RTC_GetTime()***
- ***HAL_RTC_SetDate()***
- ***HAL_RTC_GetDate()***

33.2.9 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

- ***HAL_RTC_SetAlarm()***
- ***HAL_RTC_SetAlarm_IT()***
- ***HAL_RTC_GetAlarm()***
- ***HAL_RTC_DeactivateAlarm()***
- ***HAL_RTC_AlarmIRQHandler()***
- ***HAL_RTC_AlarmAEventCallback()***
- ***HAL_RTC_PollForAlarmAEvent()***

33.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state
- ***HAL_RTC_GetState()***

33.2.11 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization
- ***HAL_RTC_WaitForSynchro()***

33.2.12 HAL_RTC_Init

Function Name	**HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)**
Function Description	Initializes the RTC peripheral.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• HAL status

33.2.13 HAL_RTC_DeInit

Function Name	**HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)**
Function Description	DeInitializes the RTC peripheral.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• HAL status
Notes	• This function does not reset the RTC Backup Data registers.

33.2.14 HAL_RTC_MspInit

Function Name	**void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)**
Function Description	Initializes the RTC MSP.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

33.2.15 HAL_RTC_MspDeInit

Function Name	**void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)**
Function Description	DeInitializes the RTC MSP.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

33.2.16 HAL_RTC_SetTime

Function Name	**HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)**
Function Description	Sets RTC current time.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **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

33.2.17 HAL_RTC_GetTime

Function Name	**HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)**
Function Description	Gets RTC current time.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **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

33.2.18 HAL_RTC_SetDate

Function Name	**HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)**
Function Description	Sets RTC current date.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **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

33.2.19 HAL_RTC_GetDate

Function Name	**HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)**
Function Description	Gets RTC current date.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **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

33.2.20 HAL_RTC_SetAlarm

Function Name	**HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)**
Function Description	Sets the specified RTC Alarm.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **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

33.2.21 HAL_RTC_SetAlarm_IT

Function Name	**HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)**
Function Description	Sets the specified RTC Alarm with Interrupt.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **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 HAL_RTC_SetTime() must be called before enabling the Alarm feature.

33.2.22 HAL_RTC_GetAlarm

Function Name	**HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)**
Function Description	Gets the RTC Alarm value and masks.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **sAlarm:** Pointer to Date structure • **Alarm:** Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: Alarm • **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

33.2.23 HAL_RTC_DeactivateAlarm

Function Name	**HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)**
Function Description	Deactive the specified RTC Alarm.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **Alarm:** Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmA
Return values	• HAL status

33.2.24 HAL_RTC_AlarmIRQHandler

Function Name	**void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)**
Function Description	This function handles Alarm interrupt request.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

33.2.25 HAL_RTC_AlarmAEventCallback

Function Name	**void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Alarm A callback.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

33.2.26 HAL_RTC_PollForAlarmAEvent

Function Name	**HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function Description	This function handles AlarmA Polling request.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **Timeout:** Timeout duration
Return values	• HAL status

33.2.27 HAL_RTC_GetState

Function Name	**HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)**
Function Description	Returns the RTC state.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• HAL state

33.2.28 HAL_RTC_WaitForSynchro

Function Name	**HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)**
Function Description	Waits until the RTC registers (RTC_CNT, RTC_ALR and RTC_PRL) are synchronized with RTC APB clock.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• HAL status
Notes	• This function must be called before any read operation after an APB reset or an APB clock stop.

33.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

33.3.1 RTC

RTC

Alarms Definitions

RTC_ALARM_A Specify alarm ID (mainly for legacy purposes)

Automatic calculation of prediv for 1sec timebase

RTC_AUTO_1_SECOND

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_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

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

Return value:

- None:

__HAL_RTC_ALARM_GET_IT_SOURCE

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be checked This parameter can be:
  - RTC_IT_ALRA: Alarm A 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 be enabled or disabled. This parameter can be:
  – RTC_FLAG_ALRAF

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

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 be enabled or disabled. This parameter can be:
  – RTC_FLAG_ALRAF

Return value:

- None:

__HAL_RTC_ALARM_EXTI_ENABLE_IT

Description:

- Enable interrupt on ALARM Exti Line 17.

	Return value: • None.:
__HAL_RTC_ALARM_EXTI_DISABLE_IT	**Description:** • Disable interrupt on ALARM Exti Line 17. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_ENABLE_EVENT	**Description:** • Enable event on ALARM Exti Line 17. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_DISABLE_EVENT	**Description:** • Disable event on ALARM Exti Line 17. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE	**Description:** • ALARM EXTI line configuration: set falling edge trigger. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE	**Description:** • Disable the ALARM Extended Interrupt Falling Trigger. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE	**Description:** • ALARM EXTI line configuration: set rising edge trigger. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE	**Description:** • Disable the ALARM Extended Interrupt Rising Trigger. **Return value:**

	• None.:
__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE	**Description:** • ALARM EXTI line configuration: set rising & falling edge trigger. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE	**Description:** • Disable the ALARM Extended Interrupt Rising & Falling Trigger. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_GET_FLAG	**Description:** • Check whether the specified ALARM EXTI interrupt flag is set or not. **Return value:** • EXTI: ALARM Line Status.
__HAL_RTC_ALARM_EXTI_CLEAR_FLAG	**Description:** • Clear the ALARM EXTI flag. **Return value:** • None.:
__HAL_RTC_ALARM_EXTI_GENERATE_SWIT	**Description:** • Generate a Software interrupt on selected EXTI line. **Return value:** • None.:

RTC EXTI Line event

RTC_EXTI_LINE_ALARM_EVENT	External interrupt line 17 Connected to the RTC Alarm event

Flags Definitions

RTC_FLAG_RTOFF	RTC Operation OFF flag
RTC_FLAG_RSF	Registers Synchronized flag
RTC_FLAG_OW	Overflow flag
RTC_FLAG_ALRAF	Alarm flag
RTC_FLAG_SEC	Second flag

RTC_FLAG_TAMP1F	Tamper Interrupt Flag

Input Parameter Format

RTC_FORMAT_BIN

RTC_FORMAT_BCD

Interrupts Definitions

RTC_IT_OW	Overflow interrupt
RTC_IT_ALRA	Alarm interrupt
RTC_IT_SEC	Second interrupt
RTC_IT_TAMP1	TAMPER Pin interrupt enable

Month Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

Output source to output on the Tamper pin

RTC_OUTPUTSOURCE_NONE	No output on the TAMPER pin
RTC_OUTPUTSOURCE_CALIBCLOCK	RTC clock with a frequency divided by 64 on the TAMPER pin
RTC_OUTPUTSOURCE_ALARM	Alarm pulse signal on the TAMPER pin
RTC_OUTPUTSOURCE_SECOND	Second pulse signal on the TAMPER pin

RTC Private Constants

RTC_ALARM_RESETVALUE_REGISTER

RTC_ALARM_RESETVALUE

RTC Private Macros

IS_RTC_ASYNCH_PREDIV

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_ALARM

IS_RTC_CALIB_OUTPUT

Default Timeout Value

RTC_TIMEOUT_VALUE

WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

34 HAL RTC Extension Driver

34.1 RTCEx Firmware driver registers structures

34.1.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the stm32f1xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Trigger***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper*** Specifies the Tamper Pin. This parameter can be a value of ***RTCEx_Tamper_Pins_Definitions***
- ***uint32_t RTC_TamperTypeDef::Trigger*** Specifies the Tamper Trigger. This parameter can be a value of ***RTCEx_Tamper_Trigger_Definitions***

34.2 RTCEx Firmware driver API description

The following section lists the various functions of the RTCEx library.

34.2.1 RTC Tamper functions

This section provides functions allowing to configure Tamper feature

- ***HAL_RTCEx_SetTamper()***
- ***HAL_RTCEx_SetTamper_IT()***
- ***HAL_RTCEx_DeactivateTamper()***
- ***HAL_RTCEx_TamperIRQHandler()***
- ***HAL_RTCEx_Tamper1EventCallback()***
- ***HAL_RTCEx_PollForTamper1Event()***

34.2.2 RTC Second functions

This section provides functions implementing second interupt handlers

- ***HAL_RTCEx_SetSecond_IT()***
- ***HAL_RTCEx_DeactivateSecond()***
- ***HAL_RTCEx_RTCIRQHandler()***
- ***HAL_RTCEx_RTCEventCallback()***
- ***HAL_RTCEx_RTCEventErrorCallback()***

34.2.3 Extension Peripheral Control functions

This subsection provides functions allowing to

- Writes a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Sets the Smooth calibration parameters.
- ***HAL_RTCEx_BKUPWrite()***
- ***HAL_RTCEx_BKUPRead()***
- ***HAL_RTCEx_SetSmoothCalib()***

34.2.4 HAL_RTCEx_SetTamper

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function Description	Sets Tamper.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **sTamper:** Pointer to Tamper Structure.
Return values	• HAL status
Notes	• By calling this API we disable the tamper interrupt for all tampers. • Tamper can be enabled only if ASOE and CCO bit are reset

34.2.5 HAL_RTCEx_SetTamper_IT

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function Description	Sets Tamper with interrupt.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **sTamper:** Pointer to RTC Tamper.
Return values	• HAL status
Notes	• By calling this API we force the tamper interrupt for all tampers. • Tamper can be enabled only if ASOE and CCO bit are reset

34.2.6 HAL_RTCEx_DeactivateTamper

Function Name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)**
Function Description	Deactivates Tamper.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **Tamper:** Selected tamper pin. This parameter can be a value of Tamper Pins Definitions
Return values	• HAL status

34.2.7 HAL_RTCEx_TamperIRQHandler

Function Name	**void HAL_RTCEx_TamperIRQHandler (RTC_HandleTypeDef * hrtc)**
Function Description	This function handles Tamper interrupt request.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

34.2.8 HAL_RTCEx_Tamper1EventCallback

Function Name	**void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Tamper 1 callback.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

34.2.9 HAL_RTCEx_PollForTamper1Event

Function Name	**HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function Description	This function handles Tamper1 Polling.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **Timeout:** Timeout duration
Return values	• HAL status

34.2.10 HAL_RTCEx_SetSecond_IT

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetSecond_IT (RTC_HandleTypeDef * hrtc)**
Function Description	Sets Interrupt for second.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• HAL status

34.2.11 HAL_RTCEx_DeactivateSecond

Function Name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateSecond**

	(RTC_HandleTypeDef * hrtc)
Function Description	Deactivates Second.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• HAL status

34.2.12 HAL_RTCEx_RTCIRQHandler

Function Name	**void HAL_RTCEx_RTCIRQHandler (RTC_HandleTypeDef * hrtc)**
Function Description	This function handles second interrupt request.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

34.2.13 HAL_RTCEx_RTCEventCallback

Function Name	**void HAL_RTCEx_RTCEventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Second event callback.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

34.2.14 HAL_RTCEx_RTCEventErrorCallback

Function Name	**void HAL_RTCEx_RTCEventErrorCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Second event error callback.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	• None

34.2.15 HAL_RTCEx_BKUPWrite

Function Name	**void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)**
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **BackupRegister:** RTC Backup data Register number. This

	parameter can be: RTC_BKP_DRx where x can be from 1 to 10 (or 42) to specify the register (depending devices). • **Data:** Data to be written in the specified RTC Backup data register.
Return values	• None

34.2.16 HAL_RTCEx_BKUPRead

Function Name	**uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)**
Function Description	Reads data from the specified RTC Backup data Register.
Parameters	• **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 1 to 10 (or 42) to specify the register (depending devices).
Return values	• Read value

34.2.17 HAL_RTCEx_SetSmoothCalib

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmouthCalibMinusPulsesValue)**
Function Description	Sets the Smooth calibration parameters.
Parameters	• **hrtc:** RTC handle • **SmoothCalibPeriod:** Not used (only present for compatibility with another families) • **SmoothCalibPlusPulses:** Not used (only present for compatibility with another families) • **SmouthCalibMinusPulsesValue:** specifies the RTC Clock Calibration value. This parameter must be a number between 0 and 0x7F.
Return values	• HAL status

34.3 RTCEx Firmware driver defines

The following section lists the various define and macros of the module.

34.3.1 RTCEx

RTCEx

Alias define maintained for legacy

HAL_RTCEx_TamperTimeStampIRQHandler

Backup Registers Definitions

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_BKP_DR32

RTC_BKP_DR33

RTC_BKP_DR34

RTC_BKP_DR35

RTC_BKP_DR36

RTC_BKP_DR37

RTC_BKP_DR38

RTC_BKP_DR39

RTC_BKP_DR40

RTC_BKP_DR41

RTC_BKP_DR42

RTCEx Exported Macros

__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_TAMP1: Tamper A 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_TAMP1: Tamper A interrupt

Return value:

- None:

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be checked. This parameter can be:

– RTC_IT_TAMP1

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 sources to be enabled or disabled. This parameter can be:
  – RTC_FLAG_TAMP1F

Return value:

- None:

__HAL_RTC_TAMPER_GET_IT

Description:

- Get the selected RTC Tamper's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be checked. This parameter can be:
  – RTC_IT_TAMP1

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 be enabled or disabled. This parameter can be:
  – RTC_FLAG_TAMP1F

Return value:

- None:

__HAL_RTC_SECOND_ENABLE_IT

Description:

- Enable the RTC Second interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Second interrupt sources to be enabled This parameter can be any combination of the following values:
  - RTC_IT_SEC: Second A interrupt

Return value:

- None:

__HAL_RTC_SECOND_DISABLE_IT

Description:

- Disable the RTC Second interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Second interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC_IT_SEC: Second A interrupt

Return value:

- None:

__HAL_RTC_SECOND_GET_IT_SOURCE

Description:

- Check whether the specified RTC Second interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Second interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_SEC: Second A interrupt

Return value:

- None:

__HAL_RTC_SECOND_GET_FLAG

Description:

- Get the selected RTC Second's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC

Second Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_SEC

Return value:

- None:

__HAL_RTC_SECOND_CLEAR_FLAG

Description:

- Clear the RTC Second's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Second Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_SEC

Return value:

- None:

__HAL_RTC_OVERFLOW_ENABLE_IT

Description:

- Enable the RTC Overflow interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Overflow interrupt sources to be enabled This parameter can be any combination of the following values:
  - RTC_IT_OW: Overflow A interrupt

Return value:

- None:

__HAL_RTC_OVERFLOW_DISABLE_IT

Description:

- Disable the RTC Overflow interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Overflow interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC_IT_OW: Overflow A interrupt

Return value:

- None:

__HAL_RTC_OVERFLOW_GET_IT_SOURCE

Description:

- Check whether the specified RTC Overflow interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Overflow interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_OW: Overflow A interrupt

Return value:

- None:

__HAL_RTC_OVERFLOW_GET_FLAG

Description:

- Get the selected RTC Overflow's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Overflow Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_OW

Return value:

- None:

__HAL_RTC_OVERFLOW_CLEAR_FLAG

Description:

- Clear the RTC Overflow's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Overflow Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_OW

Return value:

- None:

Private macros to check input parameters

IS_RTC_TAMPER

IS_RTC_TAMPER_TRIGGER

IS_RTC_BKP

IS_RTC_SMOOTH_CALIB_MINUS

Tamper Pins Definitions

RTC_TAMPER_1	Select tamper to be enabled (mainly for legacy purposes)

Tamper Trigger Definitions

RTC_TAMPERTRIGGER_LOWLEVEL	A high level on the TAMPER pin resets all data backup registers (if TPE bit is set)
RTC_TAMPERTRIGGER_HIGHLEVEL	A low level on the TAMPER pin resets all data backup registers (if TPE bit is set)

35 HAL SD Generic Driver

35.1 SD Firmware driver registers structures

35.1.1 SD_HandleTypeDef

SD_HandleTypeDef is defined in the stm32f1xx_hal_sd.h

Data Fields

- ***SD_TypeDef * Instance***
- ***SD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***uint32_t CardType***
- ***uint32_t RCA***
- ***uint32_t CSD***
- ***uint32_t CID***
- ***__IO uint32_t SdTransferCplt***
- ***__IO uint32_t SdTransferErr***
- ***__IO uint32_t DmaTransferCplt***
- ***__IO uint32_t SdOperation***
- ***DMA_HandleTypeDef * hdmarx***
- ***DMA_HandleTypeDef * hdmatx***

Field Documentation

- ***SD_TypeDef* SD_HandleTypeDef::Instance*** SDIO register base address
- ***SD_InitTypeDef SD_HandleTypeDef::Init*** SD required parameters
- ***HAL_LockTypeDef SD_HandleTypeDef::Lock*** SD locking object
- ***uint32_t SD_HandleTypeDef::CardType*** SD card type
- ***uint32_t SD_HandleTypeDef::RCA*** SD relative card address
- ***uint32_t SD_HandleTypeDef::CSD[4]*** SD card specific data table
- ***uint32_t SD_HandleTypeDef::CID[4]*** SD card identification number table
- ***__IO uint32_t SD_HandleTypeDef::SdTransferCplt*** SD transfer complete flag in non blocking mode
- ***__IO uint32_t SD_HandleTypeDef::SdTransferErr*** SD transfer error flag in non blocking mode
- ***__IO uint32_t SD_HandleTypeDef::DmaTransferCplt*** SD DMA transfer complete flag
- ***__IO uint32_t SD_HandleTypeDef::SdOperation*** SD transfer operation (read/write)
- ***DMA_HandleTypeDef* SD_HandleTypeDef::hdmarx*** SD Rx DMA handle parameters
- ***DMA_HandleTypeDef* SD_HandleTypeDef::hdmatx*** SD Tx DMA handle parameters

35.1.2 HAL_SD_CSDTypedef

HAL_SD_CSDTypedef is defined in the stm32f1xx_hal_sd.h

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_SD_CSDTypedef::CSDStruct*** CSD structure
- ***__IO uint8_t HAL_SD_CSDTypedef::SysSpecVersion*** System specification version
- ***__IO uint8_t HAL_SD_CSDTypedef::Reserved1*** Reserved
- ***__IO uint8_t HAL_SD_CSDTypedef::TAAC*** Data read access time 1
- ***__IO uint8_t HAL_SD_CSDTypedef::NSAC*** Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_SD_CSDTypedef::MaxBusClkFrec*** Max. bus clock frequency
- ***__IO uint16_t HAL_SD_CSDTypedef::CardComdClasses*** Card command classes
- ***__IO uint8_t HAL_SD_CSDTypedef::RdBlockLen*** Max. read data block length
- ***__IO uint8_t HAL_SD_CSDTypedef::PartBlockRead*** Partial blocks for read allowed
- ***__IO uint8_t HAL_SD_CSDTypedef::WrBlockMisalign*** Write block misalignment
- ***__IO uint8_t HAL_SD_CSDTypedef::RdBlockMisalign*** Read block misalignment
- ***__IO uint8_t HAL_SD_CSDTypedef::DSRImpl*** DSR implemented

- ***__IO uint8_t HAL_SD_CSDTypedef::Reserved2*** Reserved
- ***__IO uint32_t HAL_SD_CSDTypedef::DeviceSize*** Device Size
- ***__IO uint8_t HAL_SD_CSDTypedef::MaxRdCurrentVDDMin*** Max. read current @ VDD min
- ***__IO uint8_t HAL_SD_CSDTypedef::MaxRdCurrentVDDMax*** Max. read current @ VDD max
- ***__IO uint8_t HAL_SD_CSDTypedef::MaxWrCurrentVDDMin*** Max. write current @ VDD min
- ***__IO uint8_t HAL_SD_CSDTypedef::MaxWrCurrentVDDMax*** Max. write current @ VDD max
- ***__IO uint8_t HAL_SD_CSDTypedef::DeviceSizeMul*** Device size multiplier
- ***__IO uint8_t HAL_SD_CSDTypedef::EraseGrSize*** Erase group size
- ***__IO uint8_t HAL_SD_CSDTypedef::EraseGrMul*** Erase group size multiplier
- ***__IO uint8_t HAL_SD_CSDTypedef::WrProtectGrSize*** Write protect group size
- ***__IO uint8_t HAL_SD_CSDTypedef::WrProtectGrEnable*** Write protect group enable
- ***__IO uint8_t HAL_SD_CSDTypedef::ManDeflECC*** Manufacturer default ECC
- ***__IO uint8_t HAL_SD_CSDTypedef::WrSpeedFact*** Write speed factor
- ***__IO uint8_t HAL_SD_CSDTypedef::MaxWrBlockLen*** Max. write data block length
- ***__IO uint8_t HAL_SD_CSDTypedef::WriteBlockPaPartial*** Partial blocks for write allowed
- ***__IO uint8_t HAL_SD_CSDTypedef::Reserved3*** Reserved
- ***__IO uint8_t HAL_SD_CSDTypedef::ContentProtectAppli*** Content protection application
- ***__IO uint8_t HAL_SD_CSDTypedef::FileFormatGrouop*** File format group
- ***__IO uint8_t HAL_SD_CSDTypedef::CopyFlag*** Copy flag (OTP)
- ***__IO uint8_t HAL_SD_CSDTypedef::PermWrProtect*** Permanent write protection
- ***__IO uint8_t HAL_SD_CSDTypedef::TempWrProtect*** Temporary write protection
- ***__IO uint8_t HAL_SD_CSDTypedef::FileFormat*** File format
- ***__IO uint8_t HAL_SD_CSDTypedef::ECC*** ECC code
- ***__IO uint8_t HAL_SD_CSDTypedef::CSD_CRC*** CSD CRC
- ***__IO uint8_t HAL_SD_CSDTypedef::Reserved4*** Always 1

35.1.3 HAL_SD_CIDTypedef

HAL_SD_CIDTypedef is defined in the stm32f1xx_hal_sd.h

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_CIDTypedef::ManufacturerID*** Manufacturer ID

- ***__IO uint16_t HAL_SD_CIDTypedef::OEM_AppliID*** OEM/Application ID
- ***__IO uint32_t HAL_SD_CIDTypedef::ProdName1*** Product Name part1
- ***__IO uint8_t HAL_SD_CIDTypedef::ProdName2*** Product Name part2
- ***__IO uint8_t HAL_SD_CIDTypedef::ProdRev*** Product Revision
- ***__IO uint32_t HAL_SD_CIDTypedef::ProdSN*** Product Serial Number
- ***__IO uint8_t HAL_SD_CIDTypedef::Reserved1*** Reserved1
- ***__IO uint16_t HAL_SD_CIDTypedef::ManufactDate*** Manufacturing Date
- ***__IO uint8_t HAL_SD_CIDTypedef::CID_CRC*** CID CRC
- ***__IO uint8_t HAL_SD_CIDTypedef::Reserved2*** Always 1

35.1.4 HAL_SD_CardStatusTypedef

HAL_SD_CardStatusTypedef is defined in the stm32f1xx_hal_sd.h

Data Fields

- ***__IO uint8_t DAT_BUS_WIDTH***
- ***__IO uint8_t SECURED_MODE***
- ***__IO uint16_t SD_CARD_TYPE***
- ***__IO uint32_t SIZE_OF_PROTECTED_AREA***
- ***__IO uint8_t SPEED_CLASS***
- ***__IO uint8_t PERFORMANCE_MOVE***
- ***__IO uint8_t AU_SIZE***
- ***__IO uint16_t ERASE_SIZE***
- ***__IO uint8_t ERASE_TIMEOUT***
- ***__IO uint8_t ERASE_OFFSET***

Field Documentation

- ***__IO uint8_t HAL_SD_CardStatusTypedef::DAT_BUS_WIDTH*** Shows the currently defined data bus width
- ***__IO uint8_t HAL_SD_CardStatusTypedef::SECURED_MODE*** Card is in secured mode of operation
- ***__IO uint16_t HAL_SD_CardStatusTypedef::SD_CARD_TYPE*** Carries information about card type
- ***__IO uint32_t HAL_SD_CardStatusTypedef::SIZE_OF_PROTECTED_AREA*** Carries information about the capacity of protected area
- ***__IO uint8_t HAL_SD_CardStatusTypedef::SPEED_CLASS*** Carries information about the speed class of the card
- ***__IO uint8_t HAL_SD_CardStatusTypedef::PERFORMANCE_MOVE*** Carries information about the card's performance move
- ***__IO uint8_t HAL_SD_CardStatusTypedef::AU_SIZE*** Carries information about the card's allocation unit size
- ***__IO uint16_t HAL_SD_CardStatusTypedef::ERASE_SIZE*** Determines the number of AUs to be erased in one operation
- ***__IO uint8_t HAL_SD_CardStatusTypedef::ERASE_TIMEOUT*** Determines the timeout for any number of AU erase
- ***__IO uint8_t HAL_SD_CardStatusTypedef::ERASE_OFFSET*** Carries information about the erase offset

35.1.5 HAL_SD_CardInfoTypedef

HAL_SD_CardInfoTypedef is defined in the stm32f1xx_hal_sd.h

Data Fields

- ***HAL_SD_CSDTypedef SD_csd***
- ***HAL_SD_CIDTypedef SD_cid***
- ***uint64_t CardCapacity***
- ***uint32_t CardBlockSize***
- ***uint16_t RCA***
- ***uint8_t CardType***

Field Documentation

- ***HAL_SD_CSDTypedef HAL_SD_CardInfoTypedef::SD_csd*** SD card specific data register
- ***HAL_SD_CIDTypedef HAL_SD_CardInfoTypedef::SD_cid*** SD card identification number register
- ***uint64_t HAL_SD_CardInfoTypedef::CardCapacity*** Card capacity
- ***uint32_t HAL_SD_CardInfoTypedef::CardBlockSize*** Card block size
- ***uint16_t HAL_SD_CardInfoTypedef::RCA*** SD relative card address
- ***uint8_t HAL_SD_CardInfoTypedef::CardType*** SD card type

35.2 SD Firmware driver API description

The following section lists the various functions of the SD library.

35.2.1 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 (SDIO 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 SDIO memories which uses the HAL SDIO driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDIO low level resources by implement the HAL_SD_MspInit() API:
   a. Enable the SDIO interface clock using __HAL_RCC_SDIO_CLK_ENABLE();
   b. SDIO pins configuration for SD card
      - Enable the clock for the SDIO GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDIO pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. DMA Configuration if you need to use DMA process (HAL_SD_ReadBlocks_DMA() and HAL_SD_WriteBlocks_DMA() APIs).
      - Enable the DMAx interface clock using __HAL_RCC_DMAx_CLK_ENABLE();
      - Configure the DMA using the function HAL_DMA_Init() with predeclared and filled.
   d. NVIC configuration if you need to use interrupt process when using DMA transfer.

   - Configure the SDIO and DMA interrupt priorities using functions HAL_NVIC_SetPriority(); DMA priority is superior to SDIO's priority
   - Enable the NVIC DMA and SDIO IRQs using function HAL_NVIC_EnableIRQ()
   - SDIO interrupts are managed using the macros __HAL_SD_SDIO_ENABLE_IT() and __HAL_SD_SDIO_DISABLE_IT() inside the communication process.
   - SDIO interrupts pending bits are managed using the macros __HAL_SD_SDIO_GET_IT() and __HAL_SD_SDIO_CLEAR_IT()
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 (SDIO_CK) is computed as follows: SDIO_CK = SDIOCLK / (ClockDiv + 2) In initialization mode and according to the SD Card standard, make sure that the SDIO_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. These information are stored in SD handle structure in case of future use.
3. Configure the SD Card Data transfer frequency. The card transfer frequency is set to SDIOCLK / (SDIO_TRANSFER_CLK_DIV + 2). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDIO_CK) is computed as follows: SDIO_CK = SDIOCLK / (ClockDiv + 2) In transfer mode and according to the SD Card standard, make sure that the SDIO_CK frequency doesn't exceed 25MHz and 50MHz 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. After this, you have to call the function HAL_SD_CheckReadOperation(), to insure that the read transfer is done correctly in both DMA and SD sides.

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. After this, you have to call the function HAL_SD_CheckWriteOperation(), to insure that the write transfer is done correctly in both DMA and SD sides.

SD card status

- At any time, you can check the SD Card status and get the SD card state by using the HAL_SD_GetStatus() function. This function checks first if the SD card is still connected and then get the internal SD Card transfer state.
- You can also get the SD card SD Status register by using the HAL_SD_SendSDStatus() function.

SD HAL driver macros list

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

- __HAL_SD_SDIO_ENABLE : Enable the SD device
- __HAL_SD_SDIO_DISABLE : Disable the SD device
- __HAL_SD_SDIO_DMA_ENABLE: Enable the SDIO DMA transfer
- __HAL_SD_SDIO_DMA_DISABLE: Disable the SDIO DMA transfer
- __HAL_SD_SDIO_ENABLE_IT: Enable the SD device interrupt
- __HAL_SD_SDIO_DISABLE_IT: Disable the SD device interrupt
- __HAL_SD_SDIO_GET_FLAG:Check whether the specified SD flag is set or not
- __HAL_SD_SDIO_CLEAR_FLAG: Clear the SD's pending flags You can refer to the SD HAL driver header file for more useful macros

35.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

- ***HAL_SD_Init()***
- ***HAL_SD_DeInit()***
- ***HAL_SD_MspInit()***
- ***HAL_SD_MspDeInit()***

35.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

- ***HAL_SD_ReadBlocks()***
- ***HAL_SD_WriteBlocks()***
- ***HAL_SD_ReadBlocks_DMA()***
- ***HAL_SD_WriteBlocks_DMA()***
- ***HAL_SD_CheckReadOperation()***
- ***HAL_SD_CheckWriteOperation()***
- ***HAL_SD_Erase()***

- *HAL_SD_IRQHandler()*
- *HAL_SD_XferCpltCallback()*
- *HAL_SD_XferErrorCallback()*
- *HAL_SD_DMA_RxCpltCallback()*
- *HAL_SD_DMA_RxErrorCallback()*
- *HAL_SD_DMA_TxCpltCallback()*
- *HAL_SD_DMA_TxErrorCallback()*

35.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations.

- *HAL_SD_Get_CardInfo()*
- *HAL_SD_WideBusOperation_Config()*
- *HAL_SD_StopTransfer()*
- *HAL_SD_HighSpeed()*

35.2.5 Peripheral State functions

This subsection permits to get in runtime the status of the peripheral and the data flow.

- *HAL_SD_SendSDStatus()*
- *HAL_SD_GetStatus()*
- *HAL_SD_GetCardStatus()*

35.2.6 HAL_SD_Init

Function Name	**HAL_SD_ErrorTypedef HAL_SD_Init (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypedef * SDCardInfo)**
Function Description	Initializes the SD card according to the specified parameters in the SD_HandleTypeDef and create the associated handle.
Parameters	• **hsd:** SD handle • **SDCardInfo:** HAL_SD_CardInfoTypedef structure for SD card information
Return values	• HAL SD error state

35.2.7 HAL_SD_DeInit

Function Name	**HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)**
Function Description	De-Initializes the SD card.
Parameters	• **hsd:** SD handle
Return values	• HAL status

35.2.8 HAL_SD_MspInit

Function Name	**void HAL_SD_MspInit (SD_HandleTypeDef * hsd)**
Function Description	Initializes the SD MSP.
Parameters	• **hsd:** SD handle
Return values	• None

35.2.9 HAL_SD_MspDeInit

Function Name	**void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)**
Function Description	De-Initialize SD MSP.
Parameters	• **hsd:** SD handle
Return values	• None

35.2.10 HAL_SD_ReadBlocks

Function Name	**HAL_SD_ErrorTypedef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)**
Function Description	Reads block(s) from a specified address in a card.
Parameters	• **hsd:** SD handle • **pReadBuffer:** pointer to the buffer that will contain the received data • **ReadAddr:** Address from where data is to be read • **BlockSize:** SD card Data block size (in bytes) This parameter should be 512 • **NumberOfBlocks:** Number of SD blocks to read
Return values	• SD Card error state

35.2.11 HAL_SD_WriteBlocks

Function Name	**HAL_SD_ErrorTypedef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)**
Function Description	Allows to write block(s) to a specified address in a card.
Parameters	• **hsd:** SD handle • **pWriteBuffer:** pointer to the buffer that will contain the data to transmit • **WriteAddr:** Address from where data is to be written • **BlockSize:** SD card Data block size (in bytes) This parameter should be 512. • **NumberOfBlocks:** Number of SD blocks to write
Return values	• SD Card error state

35.2.12 HAL_SD_ReadBlocks_DMA

Function Name	**HAL_SD_ErrorTypedef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)**
Function Description	Reads block(s) from a specified address in a card.
Parameters	• **hsd:** SD handle • **pReadBuffer:** Pointer to the buffer that will contain the received data • **ReadAddr:** Address from where data is to be read • **BlockSize:** SD card Data block size • **NumberOfBlocks:** Number of blocks to read.
Return values	• SD Card error state
Notes	• This API should be followed by the function HAL_SD_CheckReadOperation() to check the completion of the read process • BlockSize must be 512 bytes.

35.2.13 HAL_SD_WriteBlocks_DMA

Function Name	**HAL_SD_ErrorTypedef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)**
Function Description	Writes block(s) to a specified address in a card.
Parameters	• **hsd:** SD handle • **pWriteBuffer:** pointer to the buffer that will contain the data to transmit • **WriteAddr:** Address from where data is to be read • **BlockSize:** the SD card Data block size • **NumberOfBlocks:** Number of blocks to write
Return values	• SD Card error state
Notes	• This API should be followed by the function HAL_SD_CheckWriteOperation() to check the completion of the write process (by SD current status polling). • BlockSize must be 512 bytes.

35.2.14 HAL_SD_CheckReadOperation

Function Name	**HAL_SD_ErrorTypedef HAL_SD_CheckReadOperation (SD_HandleTypeDef * hsd, uint32_t Timeout)**
Function Description	This function waits until the SD DMA data read transfer is finished.
Parameters	• **hsd:** SD handle • **Timeout:** Timeout duration
Return values	• SD Card error state

35.2.15 HAL_SD_CheckWriteOperation

Function Name	**HAL_SD_ErrorTypedef HAL_SD_CheckWriteOperation (SD_HandleTypeDef * hsd, uint32_t Timeout)**
Function Description	This function waits until the SD DMA data write transfer is finished.
Parameters	• **hsd:** SD handle • **Timeout:** Timeout duration
Return values	• SD Card error state

35.2.16 HAL_SD_Erase

Function Name	**HAL_SD_ErrorTypedef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint64_t Startaddr, uint64_t Endaddr)**
Function Description	Erases the specified memory area of the given SD card.
Parameters	• **hsd:** SD handle • **Startaddr:** Start byte address • **Endaddr:** End byte address
Return values	• SD Card error state

35.2.17 HAL_SD_IRQHandler

Function Name	**void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)**
Function Description	This function handles SD card interrupt request.
Parameters	• **hsd:** SD handle
Return values	• None

35.2.18 HAL_SD_XferCpltCallback

Function Name	**void HAL_SD_XferCpltCallback (SD_HandleTypeDef * hsd)**
Function Description	SD end of transfer callback.
Parameters	• **hsd:** SD handle
Return values	• None

35.2.19 HAL_SD_XferErrorCallback

Function Name	**void HAL_SD_XferErrorCallback (SD_HandleTypeDef * hsd)**
Function Description	SD Transfer Error callback.
Parameters	• **hsd:** SD handle
Return values	• None

35.2.20 HAL_SD_DMA_RxCpltCallback

Function Name	**void HAL_SD_DMA_RxCpltCallback (DMA_HandleTypeDef * hdma)**
Function Description	SD Transfer complete Rx callback in non blocking mode.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None

35.2.21 HAL_SD_DMA_RxErrorCallback

Function Name	**void HAL_SD_DMA_RxErrorCallback (DMA_HandleTypeDef * hdma)**
Function Description	SD DMA transfer complete Rx error callback.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None

35.2.22 HAL_SD_DMA_TxCpltCallback

Function Name	**void HAL_SD_DMA_TxCpltCallback (DMA_HandleTypeDef * hdma)**
Function Description	SD Transfer complete Tx callback in non blocking mode.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None

35.2.23 HAL_SD_DMA_TxErrorCallback

Function Name	**void HAL_SD_DMA_TxErrorCallback (DMA_HandleTypeDef * hdma)**
Function Description	SD DMA transfer complete error Tx callback.
Parameters	• **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	• None

35.2.24 HAL_SD_Get_CardInfo

Function Name	**HAL_SD_ErrorTypedef HAL_SD_Get_CardInfo (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypedef * pCardInfo)**
Function Description	Returns information about specific card.
Parameters	• **hsd:** SD handle • **pCardInfo:** Pointer to a HAL_SD_CardInfoTypedef structure that contains all SD cardinformation
Return values	• SD Card error state

35.2.25 HAL_SD_WideBusOperation_Config

Function Name	**HAL_SD_ErrorTypedef HAL_SD_WideBusOperation_Config (SD_HandleTypeDef * hsd, uint32_t WideMode)**
Function Description	Enables wide bus operation for the requested card if supported by card.
Parameters	• **hsd:** SD handle • **WideMode:** Specifies the SD card wide bus mode This parameter can be one of the following values: SDIO_BUS_WIDE_8B: 8-bit data transfer (Only for MMC) SDIO_BUS_WIDE_4B: 4-bit data transfer SDIO_BUS_WIDE_1B: 1-bit data transfer
Return values	• SD Card error state

35.2.26 HAL_SD_StopTransfer

Function Name	**HAL_SD_ErrorTypedef HAL_SD_StopTransfer (SD_HandleTypeDef * hsd)**
Function Description	Aborts an ongoing data transfer.
Parameters	• **hsd:** SD handle
Return values	• SD Card error state

35.2.27 HAL_SD_HighSpeed

Function Name	**HAL_SD_ErrorTypedef HAL_SD_HighSpeed (SD_HandleTypeDef * hsd)**
Function Description	Switches the SD card to High Speed mode.
Parameters	• **hsd:** SD handle
Return values	• SD Card error state
Notes	• This operation should be followed by the configuration of PLL to have SDIOCK clock between 67 and 75 MHz

35.2.28 HAL_SD_SendSDStatus

Function Name	**HAL_SD_ErrorTypedef HAL_SD_SendSDStatus (SD_HandleTypeDef * hsd, uint32_t * pSDstatus)**
Function Description	Returns the current SD card's status.
Parameters	• **hsd:** SD handle • **pSDstatus:** Pointer to the buffer that will contain the SD card status SD Status register)
Return values	• SD Card error state

35.2.29 HAL_SD_GetStatus

Function Name	**HAL_SD_TransferStateTypedef HAL_SD_GetStatus (SD_HandleTypeDef * hsd)**
Function Description	Gets the current sd card data status.
Parameters	• **hsd:** SD handle
Return values	• Data Transfer state

35.2.30 HAL_SD_GetCardStatus

Function Name	**HAL_SD_ErrorTypedef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypedef * pCardStatus)**
Function Description	Gets the SD card status.
Parameters	• **hsd:** SD handle • **pCardStatus:** Pointer to the HAL_SD_CardStatusTypedef structure that will contain the SD card status information
Return values	• SD Card error state

35.3 SD Firmware driver defines

The following section lists the various define and macros of the module.

35.3.1 SD

SD

SD Exported Constants

SD_CMD_GO_IDLE_STATE	Resets the SD memory card.
SD_CMD_SEND_OP_COND	Sends host capacity support information and activates the card's initialization process.

SD_CMD_ALL_SEND_CID	Asks any card connected to the host to send the CID numbers on the CMD line.
SD_CMD_SET_REL_ADDR	Asks the card to publish a new relative address (RCA).
SD_CMD_SET_DSR	Programs the DSR of all cards.
SD_CMD_SDIO_SEN_OP_COND	Sends host capacity support information (HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_HS_SWITCH	Checks switchable function (mode 0) and switch card function (mode 1).
SD_CMD_SEL_DESEL_CARD	Selects the card by its own relative address and gets deselected by any other address
SD_CMD_HS_SEND_EXT_CSD	Sends SD Memory Card interface condition, which includes host supply voltage information and asks the card whether card supports voltage.
SD_CMD_SEND_CSD	Addressed card sends its card specific data (CSD) on the CMD line.
SD_CMD_SEND_CID	Addressed card sends its card identification (CID) on the CMD line.
SD_CMD_READ_DAT_UNTIL_STOP	SD card doesn't support it.
SD_CMD_STOP_TRANSMISSION	Forces the card to stop transmission.
SD_CMD_SEND_STATUS	Addressed card sends its status register.
SD_CMD_HS_BUSTEST_READ	
SD_CMD_GO_INACTIVE_STATE	Sends an addressed card into the inactive state.
SD_CMD_SET_BLOCKLEN	Sets the block length (in bytes for SDSC) for all following block commands (read, write, lock). Default block length is fixed to 512 Bytes. Not effective for SDHS and SDXC.

SD_CMD_READ_SINGLE_BLOCK	Reads single block of size selected by SET_BLOCKLEN in case of SDSC, and a block of fixed 512 bytes in case of SDHC and SDXC.
SD_CMD_READ_MULT_BLOCK	Continuously transfers data blocks from card to host until interrupted by STOP_TRANSMISSION command.
SD_CMD_HS_BUSTEST_WRITE	64 bytes tuning pattern is sent for SDR50 and SDR104.
SD_CMD_WRITE_DAT_UNTIL_STOP	Speed class control command.
SD_CMD_SET_BLOCK_COUNT	Specify block count for CMD18 and CMD25.
SD_CMD_WRITE_SINGLE_BLOCK	Writes single block of size selected by SET_BLOCKLEN in case of SDSC, and a block of fixed 512 bytes in case of SDHC and SDXC.
SD_CMD_WRITE_MULT_BLOCK	Continuously writes blocks of data until a STOP_TRANSMISSION follows.
SD_CMD_PROG_CID	Reserved for manufacturers.
SD_CMD_PROG_CSD	Programming of the programmable bits of the CSD.
SD_CMD_SET_WRITE_PROT	Sets the write protection bit of the addressed group.
SD_CMD_CLR_WRITE_PROT	Clears the write protection bit of the addressed group.
SD_CMD_SEND_WRITE_PROT	Asks the card to send the status of the write protection bits.
SD_CMD_SD_ERASE_GRP_START	Sets the address of the first write block to be erased. (For SD card only).
SD_CMD_SD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased.
SD_CMD_ERASE_GRP_START	Sets the address of the first write block to be erased. Reserved for each

	command system set by switch function command (CMD6).
SD_CMD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE	Reserved for SD security applications.
SD_CMD_FAST_IO	SD card doesn't support it (Reserved).
SD_CMD_GO_IRQ_STATE	SD card doesn't support it (Reserved).
SD_CMD_LOCK_UNLOCK	Sets/resets the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
SD_CMD_APP_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
SD_CMD_GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general purpose/application specific commands.
SD_CMD_NO_CMD	
SD_CMD_APP_SD_SET_BUSWIDTH	SDIO_APP_CMD should be sent before sending these commands. (ACMD6) Defines the data bus width to be used for data transfer. The allowed data bus widths are given in SCR register.
SD_CMD_SD_APP_STAUS	(ACMD13) Sends the SD status.
SD_CMD_SD_APP_SEND_NUM_WRITE_BLOCKS	(ACMD22) Sends the number of the written (without errors) write blocks. Responds with 32bit+CRC data block.
SD_CMD_SD_APP_OP_COND	(ACMD41) Sends host capacity support information

	(HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_SD_APP_SET_CLR_CARD_DETECT	(ACMD42) Connects/Disconnects the 50 KOhm pull-up resistor on CD/DAT3 (pin 1) of the card.
SD_CMD_SD_APP_SEND_SCR	Reads the SD Configuration Register (SCR).
SD_CMD_SDIO_RW_DIRECT	For SD I/O card only, reserved for security specification.
SD_CMD_SDIO_RW_EXTENDED	For SD I/O card only, reserved for security specification.
SD_CMD_SD_APP_GET_MKB	SD_CMD_APP_CMD should be sent before sending these commands. For SD card only
SD_CMD_SD_APP_GET_MID	For SD card only
SD_CMD_SD_APP_SET_CER_RN1	For SD card only
SD_CMD_SD_APP_GET_CER_RN2	For SD card only
SD_CMD_SD_APP_SET_CER_RES2	For SD card only
SD_CMD_SD_APP_GET_CER_RES1	For SD card only
SD_CMD_SD_APP_SECURE_READ_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_ERASE	For SD card only
SD_CMD_SD_APP_CHANGE_SECURE_AREA	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MKB	For SD card only
STD_CAPACITY_SD_CARD_V1_1	
STD_CAPACITY_SD_CARD_V2_0	
HIGH_CAPACITY_SD_CARD	
MULTIMEDIA_CARD	
SECURE_DIGITAL_IO_CARD	
HIGH_SPEED_MULTIMEDIA_CARD	
SECURE_DIGITAL_IO_COMBO_CARD	
HIGH_CAPACITY_MMC_CARD	

SD Exported Macros

__HAL_SD_SDIO_ENABLE **Description:**

- Enable the SD device.

Parameters:

- __HANDLE__: SD Handle

Return value:

- None:

__HAL_SD_SDIO_DISABLE

Description:

- Disable the SD device.

Parameters:

- __HANDLE__: SD Handle

Return value:

- None:

__HAL_SD_SDIO_DMA_ENABLE

Description:

- Enable the SDIO DMA transfer.

Parameters:

- __HANDLE__: SD Handle

Return value:

- None:

__HAL_SD_SDIO_DMA_DISABLE

Description:

- Disable the SDIO DMA transfer.

Parameters:

- __HANDLE__: SD Handle

Return value:

- None:

__HAL_SD_SDIO_ENABLE_IT

Description:

- Enable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle
- __INTERRUPT__: specifies the SDIO interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO_IT_CTIMEOUT: Command response timeout interrupt
  - SDIO_IT_DTIMEOUT: Data timeout interrupt
  - SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDIO_IT_RXOVERR: Received FIFO

overrun error interrupt
  - SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDIO_IT_CMDSENT: Command sent (no response required) interrupt
  - SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
  - SDIO_IT_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
  - SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDIO_IT_CMDACT: Command transfer in progress interrupt
  - SDIO_IT_TXACT: Data transmit in progress interrupt
  - SDIO_IT_RXACT: Data receive in progress interrupt
  - SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDIO_IT_TXFIFOF: Transmit FIFO full interrupt
  - SDIO_IT_RXFIFOF: Receive FIFO full interrupt
  - SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
  - SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
  - SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
  - SDIO_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDIO_IT_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

Return value:

- None:

__HAL_SD_SDIO_DISABLE_IT

Description:

- Disable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle
- __INTERRUPT__: specifies the SDIO interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO_IT_DCRCFAIL: Data block

sent/received (CRC check failed) interrupt
- SDIO_IT_CTIMEOUT: Command response timeout interrupt
- SDIO_IT_DTIMEOUT: Data timeout interrupt
- SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
- SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
- SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDIO_IT_CMDSENT: Command sent (no response required) interrupt
- SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO_IT_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO_IT_CMDACT: Command transfer in progress interrupt
- SDIO_IT_TXACT: Data transmit in progress interrupt
- SDIO_IT_RXACT: Data receive in progress interrupt
- SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO_IT_TXFIFOF: Transmit FIFO full interrupt
- SDIO_IT_RXFIFOF: Receive FIFO full interrupt
- SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
- SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
- SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
- SDIO_IT_SDIOIT: SD I/O interrupt received interrupt
- SDIO_IT_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

Return value:

- None:

__HAL_SD_SDIO_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:
  - SDIO_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDIO_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDIO_FLAG_CTIMEOUT: Command response timeout
  - SDIO_FLAG_DTIMEOUT: Data timeout
  - SDIO_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDIO_FLAG_RXOVERR: Received FIFO overrun error
  - SDIO_FLAG_CMDREND: Command response received (CRC check passed)
  - SDIO_FLAG_CMDSENT: Command sent (no response required)
  - SDIO_FLAG_DATAEND: Data end (data counter, SDIDCOUNT, is zero)
  - SDIO_FLAG_STBITERR: Start bit not detected on all data signals in wide bus mode.
  - SDIO_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDIO_FLAG_CMDACT: Command transfer in progress
  - SDIO_FLAG_TXACT: Data transmit in progress
  - SDIO_FLAG_RXACT: Data receive in progress
  - SDIO_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDIO_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDIO_FLAG_TXFIFOF: Transmit FIFO full
  - SDIO_FLAG_RXFIFOF: Receive FIFO full
  - SDIO_FLAG_TXFIFOE: Transmit FIFO empty
  - SDIO_FLAG_RXFIFOE: Receive FIFO empty
  - SDIO_FLAG_TXDAVL: Data available in transmit FIFO
  - SDIO_FLAG_RXDAVL: Data available in receive FIFO
  - SDIO_FLAG_SDIOIT: SD I/O interrupt received
  - SDIO_FLAG_CEATAEND: CE-ATA command completion signal received for CMD61

Return value:

- The: new state of SD FLAG (SET or RESET).

__HAL_SD_SDIO_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:
  - SDIO_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDIO_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDIO_FLAG_CTIMEOUT: Command response timeout
  - SDIO_FLAG_DTIMEOUT: Data timeout
  - SDIO_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDIO_FLAG_RXOVERR: Received FIFO overrun error
  - SDIO_FLAG_CMDREND: Command response received (CRC check passed)
  - SDIO_FLAG_CMDSENT: Command sent (no response required)
  - SDIO_FLAG_DATAEND: Data end (data counter, SDIDCOUNT, is zero)
  - SDIO_FLAG_STBITERR: Start bit not detected on all data signals in wide bus mode
  - SDIO_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDIO_FLAG_SDIOIT: SD I/O interrupt received
  - SDIO_FLAG_CEATAEND: CE-ATA command completion signal received for CMD61

Return value:

- None:

__HAL_SD_SDIO_GET_IT

Description:

- Check whether the specified SD interrupt has occurred or not.

Parameters:

- __HANDLE__: SD Handle
- __INTERRUPT__: specifies the SDIO interrupt source to check. This parameter can be one of the following values:
  - SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt

- SDIO_IT_CTIMEOUT: Command response timeout interrupt
- SDIO_IT_DTIMEOUT: Data timeout interrupt
- SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
- SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
- SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDIO_IT_CMDSENT: Command sent (no response required) interrupt
- SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO_IT_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO_IT_CMDACT: Command transfer in progress interrupt
- SDIO_IT_TXACT: Data transmit in progress interrupt
- SDIO_IT_RXACT: Data receive in progress interrupt
- SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO_IT_TXFIFOF: Transmit FIFO full interrupt
- SDIO_IT_RXFIFOF: Receive FIFO full interrupt
- SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
- SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
- SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
- SDIO_IT_SDIOIT: SD I/O interrupt received interrupt
- SDIO_IT_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

Return value:

- The: new state of SD IT (SET or RESET).

__HAL_SD_SDIO_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:
  – SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  – SDIO_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  – SDIO_IT_CTIMEOUT: Command response timeout interrupt
  – SDIO_IT_DTIMEOUT: Data timeout interrupt
  – SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  – SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
  – SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
  – SDIO_IT_CMDSENT: Command sent (no response required) interrupt
  – SDIO_IT_DATAEND: Data end (data counter, SDIO_DCOUNT, is zero) interrupt
  – SDIO_IT_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
  – SDIO_IT_SDIOIT: SD I/O interrupt received interrupt
  – SDIO_IT_CEATAEND: CE-ATA command completion signal received for CMD61

Return value:

- None:

SD Exported Types

SD_InitTypeDef

SD_TypeDef

SD Private Constant

DATA_BLOCK_SIZE

SDIO_STATIC_FLAGS

SDIO_CMD0TIMEOUT

SD_OCR_ADDR_OUT_OF_RANGE

SD_OCR_ADDR_MISALIGNED

SD_OCR_BLOCK_LEN_ERR

SD_OCR_ERASE_SEQ_ERR

SD_OCR_BAD_ERASE_PARAM

SD_OCR_WRITE_PROT_VIOLATION

SD_OCR_LOCK_UNLOCK_FAILED

SD_OCR_COM_CRC_FAILED

SD_OCR_ILLEGAL_CMD

SD_OCR_CARD_ECC_FAILED

SD_OCR_CC_ERROR

SD_OCR_GENERAL_UNKNOWN_ERROR

SD_OCR_STREAM_READ_UNDERRUN

SD_OCR_STREAM_WRITE_OVERRUN

SD_OCR_CID_CSD_OVERWRIETE

SD_OCR_WP_ERASE_SKIP

SD_OCR_CARD_ECC_DISABLED

SD_OCR_ERASE_RESET

SD_OCR_AKE_SEQ_ERROR

SD_OCR_ERRORBITS

SD_R6_GENERAL_UNKNOWN_ERROR

SD_R6_ILLEGAL_CMD

SD_R6_COM_CRC_FAILED

SD_VOLTAGE_WINDOW_SD

SD_HIGH_CAPACITY

SD_STD_CAPACITY

SD_CHECK_PATTERN

SD_MAX_VOLT_TRIAL

SD_ALLZERO

SD_WIDE_BUS_SUPPORT

SD_SINGLE_BUS_SUPPORT

SD_CARD_LOCKED

SD_DATATIMEOUT

SD_0TO7BITS

SD_8TO15BITS

SD_16TO23BITS

SD_24TO31BITS

SD_MAX_DATA_LENGTH

SD_HALFFIFO

SD_HALFFIFOBYTES

SD_CCCC_LOCK_UNLOCK

SD_CCCC_WRITE_PROT

SD_CCCC_ERASE

SD_SDIO_SEND_IF_COND	SDIO_APP_CMD should be sent before sending these commands.

36 HAL SMARTCARD Generic Driver

36.1 SMARTCARD Firmware driver registers structures

36.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32f1xx_hal_smartcard.h

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 GuardTime***
- ***uint32_t NACKState***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate*** This member configures the SmartCard communication baud rate. The baud rate is computed using the following formula:
  - IntegerDivider = ((PCLKx) / (16 * (hsmartcard->Init.BaudRate)))
  - FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***uint32_t SMARTCARD_InitTypeDef::WordLength*** Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***SMARTCARD_Word_Length***
- ***uint32_t SMARTCARD_InitTypeDef::StopBits*** Specifies the number of stop bits transmitted. This parameter can be a value of ***SMARTCARD_Stop_Bits***
- ***uint32_t SMARTCARD_InitTypeDef::Parity*** Specifies the parity mode. This parameter can be a value of ***SMARTCARD_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 SMARTCARD_InitTypeDef::Mode*** Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***
- ***uint32_t SMARTCARD_InitTypeDef::CLKPolarity*** Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***
- ***uint32_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***
- ***uint32_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***

- ***uint32_t SMARTCARD_InitTypeDef::Prescaler*** Specifies the SmartCard Prescaler value used for dividing the system clock to provide the smartcard clock This parameter can be a value of ***SMARTCARD_Prescaler***
- ***uint32_t SMARTCARD_InitTypeDef::GuardTime*** Specifies the SmartCard Guard Time value in terms of number of baud clocks The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency
- ***uint32_t SMARTCARD_InitTypeDef::NACKState*** Specifies the SmartCard NACK Transmission state This parameter can be a value of ***SMARTCARD_NACK_State***

36.1.2 SMARTCARD_HandleTypeDef

SMARTCARD_HandleTypeDef is defined in the stm32f1xx_hal_smartcard.h

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* SMARTCARD_HandleTypeDef::Instance*** USART registers base address
- ***SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init*** SmartCard communication parameters
- ***uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr*** Pointer to SmartCard Tx transfer Buffer
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferSize*** SmartCard Tx Transfer size
- ***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
- ***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::State*** SmartCard communication state

- ***__IO uint32_t SMARTCARD_HandleTypeDef::ErrorCode*** SmartCard Error code

36.2 SMARTCARD Firmware driver API description

The following section lists the various functions of the SMARTCARD library.

36.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure.
2. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
    a. Enable the interface clock of the USARTx associated to the SMARTCARD.
    b. SMARTCARD pins configuration:
        – Enable the clock for the SMARTCARD GPIOs.
        – Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
    c. 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.
    d. 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 initilalized 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.
        – Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length , Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the SMARTCARD Init structure.
4. 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 customed HAL_SMARTCARD_MspInit(&hsc) API. The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.
5. 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_ENABLE: Enable the SMARTCARD peripheral
- __HAL_SMARTCARD_DISABLE: Disable the SMARTCARD peripheral
- __HAL_SMARTCARD_GET_FLAG : Check whether the specified SMARTCARD flag is set or not
- __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 the specified SMARTCARD interrupt has occurred or not

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

36.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in Smartcard mode.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

- For the Smartcard mode only these parameters can be configured as follows:
  - Baud Rate
  - Word Length => Should be 9 bits (8 bits + parity)
  - Stop Bit
  - Parity: => Should be enabled (see *Table 19: "Smartcard frame formats"*).
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes
  - Prescaler
  - GuardTime
  - NACKState: The Smartcard NACK state
- Recommended SmartCard interface configuration to get the Answer to Reset from the Card:
  - Word Length = 9 Bits
  - 1.5 Stop Bit
  - Even parity
  - BaudRate = 12096 baud
  - Tx and Rx enabled

Table 19: Smartcard frame formats

M bit	PCE bit	Smartcard frame
1	1	| SB | 8 bit data | PB | STB |

Please refer to the ISO 7816-3 specification for more details. -@- It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.

The HAL_SMARTCARD_Init() function follows the USART SmartCard configuration procedure (details for the procedure are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

- ***HAL_SMARTCARD_Init()***
- ***HAL_SMARTCARD_DeInit()***
- ***HAL_SMARTCARD_MspInit()***
- ***HAL_SMARTCARD_MspDeInit()***

36.2.3 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.

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, 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.
2. Blocking mode APIs are :
   - HAL_SMARTCARD_Transmit()
   - HAL_SMARTCARD_Receive()
3. Non Blocking mode APIs with Interrupt are :
   - HAL_SMARTCARD_Transmit_IT()
   - HAL_SMARTCARD_Receive_IT()
   - HAL_SMARTCARD_IRQHandler()
4. Non Blocking mode functions with DMA are :
   - HAL_SMARTCARD_Transmit_DMA()
   - HAL_SMARTCARD_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_SMARTCARD_TxCpltCallback()
   - HAL_SMARTCARD_RxCpltCallback()
   - HAL_SMARTCARD_ErrorCallback()

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.
  - No-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()

- ***HAL_SMARTCARD_Transmit()***
- ***HAL_SMARTCARD_Receive()***
- ***HAL_SMARTCARD_Transmit_IT()***
- ***HAL_SMARTCARD_Receive_IT()***
- ***HAL_SMARTCARD_Transmit_DMA()***
- ***HAL_SMARTCARD_Receive_DMA()***
- ***HAL_SMARTCARD_IRQHandler()***
- ***HAL_SMARTCARD_TxCpltCallback()***
- ***HAL_SMARTCARD_RxCpltCallback()***
- ***HAL_SMARTCARD_ErrorCallback()***

36.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard communication process 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() check in run-time errors that could be occurred during communication.
- ***HAL_SMARTCARD_GetState()***
- ***HAL_SMARTCARD_GetError()***

36.2.5 HAL_SMARTCARD_Init

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsc)**
Function Description	Initializes the SmartCard mode according to the specified parameters in the SMARTCARD_HandleTypeDef and create the associated handle.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• HAL status

36.2.6 HAL_SMARTCARD_DeInit

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_DeInit**

	(SMARTCARD_HandleTypeDef * hsc)
Function Description	DeInitializes the SMARTCARD peripheral.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• HAL status

36.2.7 HAL_SMARTCARD_MspInit

Function Name	**void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsc)**
Function Description	SMARTCARD MSP Init.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• None

36.2.8 HAL_SMARTCARD_MspDeInit

Function Name	**void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsc)**
Function Description	SMARTCARD MSP DeInit.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• None

36.2.9 HAL_SMARTCARD_Transmit

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Sends an amount of data in blocking mode.
Parameters	• **hsc:** 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:** Specify timeout value
Return values	• HAL status

36.2.10 HAL_SMARTCARD_Receive

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Receive an amount of data in blocking mode.
Parameters	• **hsc:** 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:** Specify timeout value
Return values	• HAL status

36.2.11 HAL_SMARTCARD_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)**
Function Description	Sends an amount of data in non-blocking mode.
Parameters	• **hsc:** 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

36.2.12 HAL_SMARTCARD_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)**
Function Description	Receives an amount of data in non-blocking mode.
Parameters	• **hsc:** 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

36.2.13 HAL_SMARTCARD_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)**
Function Description	Sends an amount of data in non-blocking mode.

Parameters	• **hsc:** 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

36.2.14 HAL_SMARTCARD_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)**
Function Description	Receive an amount of data in non-blocking mode.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be received
Return values	• HAL status
Notes	• When the SMARTCARD parity is enabled (PCE = 1) the data received contain the parity bit.

36.2.15 HAL_SMARTCARD_IRQHandler

Function Name	**void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsc)**
Function Description	This function handles SMARTCARD interrupt request.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• None

36.2.16 HAL_SMARTCARD_TxCpltCallback

Function Name	**void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsc)**
Function Description	Tx Transfer completed callback.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• None

36.2.17 HAL_SMARTCARD_RxCpltCallback

Function Name	**void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsc)**
Function Description	Rx Transfer completed callback.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• None

36.2.18 HAL_SMARTCARD_ErrorCallback

Function Name	**void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsc)**
Function Description	SMARTCARD error callback.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• None

36.2.19 HAL_SMARTCARD_GetState

Function Name	**HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsc)**
Function Description	Returns the SMARTCARD state.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• HAL state

36.2.20 HAL_SMARTCARD_GetError

Function Name	**uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsc)**
Function Description	Return the SMARTCARD error code.
Parameters	• **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	• SMARTCARD Error Code

36.3 SMARTCARD Firmware driver defines

The following section lists the various define and macros of the module.

36.3.1 SMARTCARD

SMARTCARD

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD_PHASE_2EDGE

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD_POLARITY_HIGH

SMARTCARD DMA requests

SMARTCARD_DMAREQ_TX

SMARTCARD_DMAREQ_RX

SMARTCARD Error Codes

HAL_SMARTCARD_ERROR_NONE	No error
HAL_SMARTCARD_ERROR_PE	Parity error
HAL_SMARTCARD_ERROR_NE	Noise error
HAL_SMARTCARD_ERROR_FE	frame error
HAL_SMARTCARD_ERROR_ORE	Overrun error
HAL_SMARTCARD_ERROR_DMA	DMA transfer error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle state.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:

- Flush the Smartcard DR register.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_SMARTCARD_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – 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_CLEAR_FLAG

Description:

- Clear the specified Smartcard pending flags.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  – SMARTCARD_FLAG_TC: Transmission Complete flag.
  – SMARTCARD_FLAG_RXNE: Receive data register not empty flag.

Return value:

- None:
- None:

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_SMARTCARD_CLEAR_OREFLAG	**Description:** • Clear the SMARTCARD ORE pending flag. **Parameters:** • __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device). **Return value:** • None:
__HAL_SMARTCARD_CLEAR_IDLEFLAG	**Description:** • Clear the SMARTCARD IDLE pending flag. **Parameters:** • __HANDLE__: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device). **Return value:** • None:
__HAL_SMARTCARD_ENABLE_IT	**Description:** • Enable the specified SmartCard interrupt. **Parameters:** • __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device). • __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values: – SMARTCARD_IT_TXE: Transmit Data Register empty interrupt – SMARTCARD_IT_TC: Transmission complete interrupt – 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 interrupts.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  - SMARTCARD_IT_TXE: Transmit Data Register empty interrupt
  - SMARTCARD_IT_TC: Transmission complete interrupt
  - 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)

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __IT__: specifies the SMARTCARD

interrupt source to check. This parameter can be one of the following values:
- SMARTCARD_IT_TXE: Transmit Data Register empty interrupt
- SMARTCARD_IT_TC: Transmission complete interrupt
- SMARTCARD_IT_RXNE: Receive Data register not empty interrupt
- SMARTCARD_IT_IDLE: Idle line detection interrupt
- SMARTCARD_IT_ERR: Error interrupt
- SMARTCARD_IT_PE: Parity Error interrupt

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

__HAL_SMARTCARD_ENABLE

Description:
- Enable the USART associated to the SMARTCARD Handle.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:
- None:

__HAL_SMARTCARD_DISABLE

Description:
- Disable the USART associated to the SMARTCARD Handle.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:
- None:

__HAL_SMARTCARD_DMA_REQUEST_ENA

Description:

BLE

- Enable the SmartCard DMA request.

Parameters:

- __HANDLE__: specifies the SmartCard Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __REQUEST__: specifies the SmartCard DMA request. This parameter can be one of the following values:
  – SMARTCARD_DMAREQ_TX: SmartCard DMA transmit request
  – SMARTCARD_DMAREQ_RX: SmartCard DMA receive request

Return value:

- None:

__HAL_SMARTCARD_DMA_REQUEST_DISABLE

Description:

- Disable the SmartCard DMA request.

Parameters:

- __HANDLE__: specifies the SmartCard Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __REQUEST__: specifies the SmartCard DMA request. This parameter can be one of the following values:
  – SMARTCARD_DMAREQ_TX: SmartCard DMA transmit request
  – SMARTCARD_DMAREQ_RX: SmartCard DMA receive request

Return value:

- None:

SMARTCARD Flags

SMARTCARD_FLAG_TXE

SMARTCARD_FLAG_TC

SMARTCARD_FLAG_RXNE

SMARTCARD_FLAG_IDLE

SMARTCARD_FLAG_ORE

SMARTCARD_FLAG_NE

SMARTCARD_FLAG_FE

SMARTCARD_FLAG_PE

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE

SMARTCARD_IT_TXE

SMARTCARD_IT_TC

SMARTCARD_IT_RXNE

SMARTCARD_IT_IDLE

SMARTCARD_IT_ERR

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE

SMARTCARD_LASTBIT_ENABLE

SMARTCARD Mode

SMARTCARD_MODE_RX

SMARTCARD_MODE_TX

SMARTCARD_MODE_TX_RX

SMARTCARD NACK State

SMARTCARD_NACK_ENABLE

SMARTCARD_NACK_DISABLE

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD_PARITY_ODD

SMARTCARD Prescaler

SMARTCARD_PRESCALER_SYSCLK_DIV2	SYSCLK divided by 2
SMARTCARD_PRESCALER_SYSCLK_DIV4	SYSCLK divided by 4
SMARTCARD_PRESCALER_SYSCLK_DIV6	SYSCLK divided by 6
SMARTCARD_PRESCALER_SYSCLK_DIV8	SYSCLK divided by 8
SMARTCARD_PRESCALER_SYSCLK_DIV10	SYSCLK divided by 10
SMARTCARD_PRESCALER_SYSCLK_DIV12	SYSCLK divided by 12
SMARTCARD_PRESCALER_SYSCLK_DIV14	SYSCLK divided by 14
SMARTCARD_PRESCALER_SYSCLK_DIV16	SYSCLK divided by 16
SMARTCARD_PRESCALER_SYSCLK_DIV18	SYSCLK divided by 18
SMARTCARD_PRESCALER_SYSCLK_DIV20	SYSCLK divided by 20
SMARTCARD_PRESCALER_SYSCLK_DIV22	SYSCLK divided by 22

SMARTCARD_PRESCALER_SYSCLK_DIV24	SYSCLK divided by 24
SMARTCARD_PRESCALER_SYSCLK_DIV26	SYSCLK divided by 26
SMARTCARD_PRESCALER_SYSCLK_DIV28	SYSCLK divided by 28
SMARTCARD_PRESCALER_SYSCLK_DIV30	SYSCLK divided by 30
SMARTCARD_PRESCALER_SYSCLK_DIV32	SYSCLK divided by 32
SMARTCARD_PRESCALER_SYSCLK_DIV34	SYSCLK divided by 34
SMARTCARD_PRESCALER_SYSCLK_DIV36	SYSCLK divided by 36
SMARTCARD_PRESCALER_SYSCLK_DIV38	SYSCLK divided by 38
SMARTCARD_PRESCALER_SYSCLK_DIV40	SYSCLK divided by 40
SMARTCARD_PRESCALER_SYSCLK_DIV42	SYSCLK divided by 42
SMARTCARD_PRESCALER_SYSCLK_DIV44	SYSCLK divided by 44
SMARTCARD_PRESCALER_SYSCLK_DIV46	SYSCLK divided by 46
SMARTCARD_PRESCALER_SYSCLK_DIV48	SYSCLK divided by 48
SMARTCARD_PRESCALER_SYSCLK_DIV50	SYSCLK divided by 50
SMARTCARD_PRESCALER_SYSCLK_DIV52	SYSCLK divided by 52
SMARTCARD_PRESCALER_SYSCLK_DIV54	SYSCLK divided by 54
SMARTCARD_PRESCALER_SYSCLK_DIV56	SYSCLK divided by 56
SMARTCARD_PRESCALER_SYSCLK_DIV58	SYSCLK divided by 58
SMARTCARD_PRESCALER_SYSCLK_DIV60	SYSCLK divided by 60
SMARTCARD_PRESCALER_SYSCLK_DIV62	SYSCLK divided by 62

SMARTCARD Private Macros

SMARTCARD_CR1_REG_INDEX	
SMARTCARD_CR3_REG_INDEX	
SMARTCARD_DIV	
SMARTCARD_DIVMANT	
SMARTCARD_DIVFRAQ	
SMARTCARD_BRR	
IS_SMARTCARD_BAUDRATE	The maximum Baud Rate is derived from the maximum clock on APB (i.e. 72 MHz) divided by the smallest oversampling used on the USART (i.e. 16) __BAUDRATE__: Baud rate set by the configuration function. Return : TRUE or FALSE
IS_SMARTCARD_WORD_LENGTH	
IS_SMARTCARD_STOPBITS	
IS_SMARTCARD_PARITY	
IS_SMARTCARD_MODE	
IS_SMARTCARD_POLARITY	
IS_SMARTCARD_PHASE	

IS_SMARTCARD_LASTBIT

IS_SMARTCARD_NACK_STATE

IS_SMARTCARD_PRESCALER

SMARTCARD_IT_MASK

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD_STOPBITS_1_5

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

37 HAL SPI Generic Driver

37.1 SPI Firmware driver registers structures

37.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32f1xx_hal_spi.h

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***

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 Directional 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 a number between Min_Data = 0 and Max_Data = 65535

37.1.2 __SPI_HandleTypeDef

__SPI_HandleTypeDef is defined in the stm32f1xx_hal_spi.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***void(* RxISR***
- ***void(* TxISR***
- ***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
- ***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
- ***uint16_t __SPI_HandleTypeDef::RxXferCount*** SPI Rx Transfer Counter
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx*** SPI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx*** SPI Rx DMA handle parameters
- ***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
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock*** SPI locking object
- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State*** SPI communication state
- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode*** SPI Error code

37.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

37.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit ()API:
   a. Enable the SPIx interface clock
   b. SPI pins configuration
      – Enable the clock for the SPI GPIOs
      – Configure these SPI pins as alternate function push-pull
   c. NVIC configuration if you need to use interrupt process
      – Configure the SPIx interrupt priority
      – Enable the NVIC SPI IRQ handle
   d. DMA Configuration if you need to use DMA process
      – Declare a DMA_HandleTypeDef handle structure for the transmit or receive Channel
      – Enable the DMAx clock
      – Configure the DMA handle parameters
      – Configure the DMA Tx or Rx Channel
      – Associate the initilalized hdma_tx(or _rx) 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 Channel
3. Program the Mode, Direction , 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 customed HAL_SPI_MspInit() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
   a. Master 2Lines RxOnly
   b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL_SPI_DMAPause()/ HAL_SPI_DMAStop() only under the SPI callbacks

Using the HAL it is not possible to reach all supported SPI frequency with the differents SPI Modes. *Table 20: "Maximum SPI frequency for 8-bit SPI data transfers"* and *Table 21: "Maximum SPI frequency for 16-bit SPI data transfers"* summarize the maximum SPI frequency reached with data size 8bits/16bits, according to frequency used on APBx Peripheral Clock (fPCLK) used by the SPI instance.

The max SPI frequency depend on SPI data size (8bits, 16bits), SPI mode(2 Lines fullduplex, 2 lines RxOnly, 1 line TX/RX) and Process mode (Polling, IT, DMA).

1. TX/RX processes are HAL_SPI_TransmitReceive(), HAL_SPI_TransmitReceive_IT() and HAL_SPI_TransmitReceive_DMA()
2. RX processes are HAL_SPI_Receive(), HAL_SPI_Receive_IT() and HAL_SPI_Receive_DMA()
3. TX processes are HAL_SPI_Transmit(), HAL_SPI_Transmit_IT() and HAL_SPI_Transmit_DMA()

Table 20: Maximum SPI frequency for 8-bit SPI data transfers

Process	Transfer mode	2 lines, fullduplex		2 line, Rx only		1 line	
		Master	Slave	Master	Slave	Master	Slave
Tx/Rx	Polling	$f_{CPU}/8$	$f_{CPU}/8$	NA	NA	NA	NA
	Interrupt	$f_{CPU}/32$	$f_{CPU}/32$	NA	NA	NA	NA
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	NA	NA
Rx	Polling	$f_{CPU}/4$	$f_{CPU}/8$	$f_{CPU}/128$	$f_{CPU}/16$	$f_{CPU}/128$	$f_{CPU}/8$
	Interrupt	$f_{CPU}/32$	$f_{CPU}/16$	$f_{CPU}/128$	$f_{CPU}/16$	$f_{CPU}/128$	$f_{CPU}/16$
	DMA	$f_{CPU}/2$	$f_{CPU}/2$	$f_{CPU}/128$	$f_{CPU}/16$	$f_{CPU}/128$	$f_{CPU}/2$
Tx	Polling	$f_{CPU}/4$	$f_{CPU}/4$	NA	NA	$f_{CPU}/4$	$f_{CPU}/64$
	Interrupt	$f_{CPU}/8$	$f_{CPU}/16$	NA	NA	$f_{CPU}/8$	$f_{CPU}/128$
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/2$	$f_{CPU}/64$

Table 21: Maximum SPI frequency for 16-bit SPI data transfers

Process	Transfer mode	2 lines, fullduplex		2 line, Rx only		1 line	
		Master	Slave	Master	Slave	Master	Slave
Tx/Rx	Polling	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	NA	NA
	Interrupt	$f_{CPU}/16$	$f_{CPU}/16$	NA	NA	NA	NA
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	NA	NA
Rx	Polling	$f_{CPU}/2$	$f_{CPU}/4$	$f_{CPU}/64$	$f_{CPU}/8$	$f_{CPU}/64$	$f_{CPU}/4$
	Interrupt	$f_{CPU}/16$	$f_{CPU}/8$	$f_{CPU}/128$	$f_{CPU}/8$	$f_{CPU}/128$	$f_{CPU}/8$
	DMA	$f_{CPU}/2$	$f_{CPU}/2$	$f_{CPU}/128$	$f_{CPU}/8$	$f_{CPU}/128$	$f_{CPU}/2$
Tx	Polling	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/2$	$f_{CPU}/64$
	Interrupt	$f_{CPU}/4$	$f_{CPU}/8$	NA	NA	$f_{CPU}/4$	$f_{CPU}/256$
	DMA	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/2$	$f_{CPU}/32$

37.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze 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
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx periperal.
- *HAL_SPI_Init()*
- *HAL_SPI_DeInit()*
- *HAL_SPI_MspInit()*
- *HAL_SPI_MspDeInit()*

37.2.3 IO operation functions

The SPI supports master and slave mode :

1. There are two modes of transfer:
   - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectivelly at the end of the transmit or Receive process The HAL_SPI_ErrorCallback()user callback will be executed when a communication error is detected
2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

- *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_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()*

37.2.4 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
- ***HAL_SPI_GetState()***
- ***HAL_SPI_GetError()***

37.2.5 HAL_SPI_Init

Function Name	**HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)**
Function Description	Initializes the SPI according to the specified parameters in the SPI_InitTypeDef and create the associated handle.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• HAL status

37.2.6 HAL_SPI_DeInit

Function Name	**HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)**
Function Description	DeInitializes the SPI peripheral.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• HAL status

37.2.7 HAL_SPI_MspInit

Function Name	**void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)**
Function Description	SPI MSP Init.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.8 HAL_SPI_MspDeInit

Function Name	**void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)**
Function Description	SPI MSP DeInit.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.9 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

37.2.10 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 sent • **Timeout:** Timeout duration
Return values	• HAL status

37.2.11 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 to be • **Size:** amount of data to be sent • **Timeout:** Timeout duration
Return values	• HAL status

37.2.12 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 no-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

37.2.13 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 no-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

37.2.14 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 no-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 to be • **Size:** amount of data to be sent
Return values	• HAL status

37.2.15 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 no-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

37.2.16 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 no-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.

37.2.17 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 no-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

37.2.18 HAL_SPI_DMAPause

Function Name	**HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)**
Function Description	Pauses the DMA Transfer.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	• HAL status

37.2.19 HAL_SPI_DMAResume

Function Name	**HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)**

Function Description	Resumes the DMA Transfer.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	• HAL status

37.2.20 HAL_SPI_DMAStop

Function Name	**HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)**
Function Description	Stops the DMA Transfer.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	• HAL status

37.2.21 HAL_SPI_IRQHandler

Function Name	**void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)**
Function Description	This function handles SPI interrupt request.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.22 HAL_SPI_TxCpltCallback

Function Name	**void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)**
Function Description	Tx Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.23 HAL_SPI_RxCpltCallback

Function Name	**void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)**
Function Description	Rx Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.24 HAL_SPI_TxRxCpltCallback

Function Name	**void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)**
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.25 HAL_SPI_TxHalfCpltCallback

Function Name	**void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function Description	Tx Half Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.26 HAL_SPI_RxHalfCpltCallback

Function Name	**void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function Description	Rx Half Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.27 HAL_SPI_TxRxHalfCpltCallback

Function Name	**void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)**
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.28 HAL_SPI_ErrorCallback

Function Name	**void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)**
Function Description	SPI error callbacks.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• None

37.2.29 HAL_SPI_GetState

Function Name	**HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)**
Function Description	Return the SPI state.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	• SPI state

37.2.30 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

37.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

37.3.1 SPI

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 data size

SPI_DATASIZE_8BIT

SPI_DATASIZE_16BIT

SPI Direction mode

SPI_DIRECTION_2LINES

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_DMA	DMA transfer error
HAL_SPI_ERROR_FLAG	Flag: RXNE,TXE, BSY

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  – SPI_IT_TXE: Tx buffer empty interrupt enable
  – SPI_IT_RXNE: RX buffer not empty interrupt enable
  – SPI_IT_ERR: Error interrupt enable

Return value:

- None:

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

- None:

__HAL_SPI_GET_IT_SOURCE

Description:

- Check if the specified SPI interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

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

__HAL_SPI_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXNE: Receive buffer not empty flag

- SPI_FLAG_TXE: Transmit buffer empty flag
- SPI_FLAG_CRCERR: CRC error flag
- SPI_FLAG_MODF: Mode fault flag
- SPI_FLAG_OVR: Overrun flag
- SPI_FLAG_BSY: Busy flag

Return value:

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

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

__HAL_SPI_ENABLE

Description:

- Enables the SPI.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

__HAL_SPI_DISABLE

Description:

- Disables the SPI.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

SPI Flag definition

SPI_FLAG_RXNE

SPI_FLAG_TXE

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_OVR

SPI_FLAG_BSY

SPI Interrupt configuration definition

SPI_IT_TXE

SPI_IT_RXNE

SPI_IT_ERR

SPI mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

SPI MSB LSB transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

SPI Private Constants

SPI_TIMEOUT_VALUE

SPI_INVALID_CRC_ERROR

SPI_VALID_CRC_ERROR

SPI Private Macros

IS_SPI_MODE

Description:

- Checks if SPI Mode parameter is in allowed range.

Parameters:

- __MODE__: specifies the SPI Mode. This parameter can be a value of

Return value:

- None:

IS_SPI_DIRECTION_MODE

Description:

- Checks if SPI Direction Mode parameter is in allowed range.

Parameters:

- __MODE__: specifies the SPI Direction Mode. This parameter can be a value of

Return value:

- None:

IS_SPI_DIRECTION_2LINES_OR_1LINE

Description:

- Checks if SPI Direction Mode parameter is 1 or 2 lines.

Parameters:

- __MODE__: specifies the SPI Direction Mode.

Return value:

- None:

IS_SPI_DIRECTION_2LINES

Description:

- Checks if SPI Direction Mode parameter is 2 lines.

Parameters:

- __MODE__: specifies the SPI Direction Mode.

Return value:

- None:

IS_SPI_DATASIZE

Description:

- Checks if SPI Data Size parameter is in allowed range.

Parameters:

- __DATASIZE__: specifies the SPI Data Size. This parameter can be a value of

Return value:

- None:

IS_SPI_CPOL

Description:

- Checks if SPI Serial clock steady state parameter is in allowed range.

Parameters:

- __CPOL__: specifies the SPI serial clock steady state. This parameter can be a value of

Return value:

- None:

IS_SPI_CPHA

Description:

- Checks if SPI Clock Phase parameter is in allowed range.

Parameters:

- __CPHA__: specifies the SPI Clock Phase. This parameter can be a value of

Return value:

- None:

IS_SPI_NSS

Description:

- Checks if SPI Slave select parameter is in allowed range.

Parameters:

- __NSS__: specifies the SPI Slave Slelect management parameter. This parameter can be a value of

Return value:

- None:

IS_SPI_BAUDRATE_PRESCALER

Description:

- Checks if SPI Baudrate prescaler parameter is in allowed range.

Parameters:

- __PRESCALER__: specifies the SPI Baudrate prescaler. This parameter can be a value of

Return value:

- None:

IS_SPI_FIRST_BIT

Description:

- Checks if SPI MSB LSB transmission parameter is in allowed range.

Parameters:

- __BIT__: specifies the SPI MSB LSB transmission (whether data transfer starts from MSB or LSB bit). This parameter can be a value of

Return value:

- None:

IS_SPI_TIMODE

Description:

- Checks if SPI TI mode parameter is in allowed range.

Parameters:

- __MODE__: specifies the SPI TI mode. This parameter can be a value of

Return value:

- None:

IS_SPI_CRC_CALCULATION

Description:

- Checks if SPI CRC calculation enabled state is in allowed range.

Parameters:

- __CALCULATION__: specifies the SPI CRC calculation enable state. This parameter can be a value of

Return value:

- None:

IS_SPI_CRC_POLYNOMIAL

Description:

- Checks if SPI polynomial value to be used for the CRC calculation, is in allowed range.

Parameters:

- __POLYNOMIAL__: specifies the SPI polynomial value to be used for the CRC calculation. This parameter must be a number between Min_Data = 0 and Max_Data = 65535

Return value:

- None:

SPI_1LINE_TX

Description:

- Sets the SPI transmit-only mode.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

SPI_1LINE_RX

Description:

- Sets the SPI receive-only mode.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

SPI_RESET_CRC

Description:

- Resets the CRC calculation of the SPI.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

SPI Slave Select management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI mode disable

SPI_TIMODE_DISABLE

38 HAL SRAM Generic Driver

38.1 SRAM Firmware driver registers structures

38.1.1 SRAM_HandleTypeDef

SRAM_HandleTypeDef is defined in the stm32f1xx_hal_sram.h

Data Fields

- ***FSMC_NORSRAM_TypeDef * Instance***
- ***FSMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FSMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdma***

Field Documentation

- ***FSMC_NORSRAM_TypeDef* SRAM_HandleTypeDef::Instance*** Register base address
- ***FSMC_NORSRAM_EXTENDED_TypeDef* SRAM_HandleTypeDef::Extended*** Extended mode register base address
- ***FSMC_NORSRAM_InitTypeDef SRAM_HandleTypeDef::Init*** SRAM device control configuration parameters
- ***HAL_LockTypeDef SRAM_HandleTypeDef::Lock*** SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef SRAM_HandleTypeDef::State*** SRAM device access state
- ***DMA_HandleTypeDef* SRAM_HandleTypeDef::hdma*** Pointer DMA handler

38.2 SRAM Firmware driver API description

The following section lists the various functions of the SRAM library.

38.2.1 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 FSMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FSMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example: SRAM_HandleTypeDef hsram; and:
   – Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
   – Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode

2. Declare two FSMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FSMC_NORSRAM_TimingTypeDef Timing and FSMC_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 FSMC NORSRAM interface function FSMC_NORSRAM_Init()
   c. Timing register configuration using the FSMC NORSRAM interface function FSMC_NORSRAM_Timing_Init()
   d. Extended mode Timing register configuration using the FSMC NORSRAM interface function FSMC_NORSRAM_Extended_Timing_Init()
   e. Enable the SRAM device using the macro __FSMC_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()

38.2.2 SRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

- ***HAL_SRAM_Init()***
- ***HAL_SRAM_DeInit()***
- ***HAL_SRAM_MspInit()***
- ***HAL_SRAM_MspDeInit()***
- ***HAL_SRAM_DMA_XferCpltCallback()***
- ***HAL_SRAM_DMA_XferErrorCallback()***

38.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

- ***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()***

38.2.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

- *HAL_SRAM_WriteOperation_Enable()*
- *HAL_SRAM_WriteOperation_Disable()*

38.2.5 SRAM State functions

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

- *HAL_SRAM_GetState()*

38.2.6 HAL_SRAM_Init

Function Name	**HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_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

38.2.7 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

38.2.8 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

38.2.9 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

38.2.10 HAL_SRAM_DMA_XferCpltCallback

Function Name	**void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)**
Function Description	DMA transfer complete callback.
Parameters	• **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• None

38.2.11 HAL_SRAM_DMA_XferErrorCallback

Function Name	**void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)**
Function Description	DMA transfer complete error callback.
Parameters	• **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	• None

38.2.12 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

38.2.13 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

38.2.14 HAL_SRAM_Read_16b

Function Name	**HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)**
Function Description	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

38.2.15 HAL_SRAM_Write_16b

Function Name	**HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)**
Function Description	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

38.2.16 HAL_SRAM_Read_32b

Function Name	**HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)**
Function Description	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

38.2.17 HAL_SRAM_Write_32b

Function Name	**HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)**
Function Description	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

38.2.18 HAL_SRAM_Read_DMA

Function Name	**HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)**
Function Description	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

38.2.19 HAL_SRAM_Write_DMA

Function Name	**HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)**
Function Description	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

38.2.20 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

38.2.21 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

38.2.22 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

38.3 SRAM Firmware driver defines

The following section lists the various define and macros of the module.

38.3.1 SRAM

SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE	**Description:** • Reset SRAM handle state. **Parameters:** • __HANDLE__: SRAM handle **Return value:** • None:

39 HAL TIM Generic Driver

39.1 TIM Firmware driver registers structures

39.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32f1xx_hal_tim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler*** Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_Base_InitTypeDef::CounterMode*** Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_Base_InitTypeDef::Period*** Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision*** Specifies the clock division. This parameter can be a value of ***TIM_ClockDivision***
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter*** Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned 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.

39.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32f1xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode*** Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OC_InitTypeDef::Pulse*** Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity*** Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity*** Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode*** Specifies the Fast mode state. This parameter can be a value of ***TIM_Output_Fast_State***
  Note: This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState*** Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState*** Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.

39.1.3 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the stm32f1xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode*** Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse*** Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity*** Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity*** Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState*** Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of

TIM_Output_Compare_Idle_State
Note: This parameter is valid only for TIM1 and TIM8.

- ***uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState*** Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note: This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity*** Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection*** Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter*** Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

39.1.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32f1xx_hal_tim.h

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

39.1.5 TIM_Encoder_InitTypeDef

TIM_Encoder_InitTypeDef is defined in the stm32f1xx_hal_tim.h

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

39.1.6 TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32f1xx_hal_tim.h

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

39.1.7 TIM_ClearInputConfigTypeDef

TIM_ClearInputConfigTypeDef is defined in the stm32f1xx_hal_tim.h

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState*** TIM clear Input state This parameter can be ENABLE or DISABLE
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource*** TIM clear Input sources This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity*** TIM Clear Input polarity This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler*** TIM Clear Input prescaler This parameter can be a value of ***TIM_ClearInput_Prescaler***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter*** TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

39.1.8 TIM_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32f1xx_hal_tim.h

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

39.1.9 TIM_HandleTypeDef

TIM_HandleTypeDef is defined in the stm32f1xx_hal_tim.h

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 ***TIM_DMA_Handle_index***
- ***HAL_LockTypeDef TIM_HandleTypeDef::Lock*** Locking object
- ***__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State*** TIM operation state

39.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

39.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
   - Input Capture
   - Output Compare
   - PWM generation (Edge and Center-aligned Mode)
   - One-pulse mode output

39.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
   - 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_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()

39.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.
- ***HAL_TIM_Base_Init()***
- ***HAL_TIM_Base_DeInit()***
- ***HAL_TIM_Base_MspInit()***
- ***HAL_TIM_Base_MspDeInit()***
- ***HAL_TIM_Base_Start()***
- ***HAL_TIM_Base_Stop()***
- ***HAL_TIM_Base_Start_IT()***
- ***HAL_TIM_Base_Stop_IT()***
- ***HAL_TIM_Base_Start_DMA()***
- ***HAL_TIM_Base_Stop_DMA()***

39.2.4 Time Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the Time Output Compare.
- Stop the Time Output Compare.
- Start the Time Output Compare and enable interrupt.
- Stop the Time Output Compare and disable interrupt.
- Start the Time Output Compare and enable DMA transfer.
- Stop the Time Output Compare and disable DMA transfer.
- ***HAL_TIM_OC_Init()***
- ***HAL_TIM_OC_DeInit()***
- ***HAL_TIM_OC_MspInit()***
- ***HAL_TIM_OC_MspDeInit()***
- ***HAL_TIM_OC_Start()***
- ***HAL_TIM_OC_Stop()***
- ***HAL_TIM_OC_Start_IT()***
- ***HAL_TIM_OC_Stop_IT()***
- ***HAL_TIM_OC_Start_DMA()***
- ***HAL_TIM_OC_Stop_DMA()***

39.2.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.
- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.
- ***HAL_TIM_PWM_Init()***
- ***HAL_TIM_PWM_DeInit()***
- ***HAL_TIM_PWM_MspInit()***
- ***HAL_TIM_PWM_MspDeInit()***
- ***HAL_TIM_PWM_Start()***
- ***HAL_TIM_PWM_Stop()***
- ***HAL_TIM_PWM_Start_IT()***
- ***HAL_TIM_PWM_Stop_IT()***
- ***HAL_TIM_PWM_Start_DMA()***
- ***HAL_TIM_PWM_Stop_DMA()***

39.2.6 Time Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the Time Input Capture.
- Stop the Time Input Capture.
- Start the Time Input Capture and enable interrupt.
- Stop the Time Input Capture and disable interrupt.

- Start the Time Input Capture and enable DMA transfer.
- Stop the Time Input Capture and disable DMA transfer.
- *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()*

39.2.7 Time One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the Time One Pulse.
- Stop the Time One Pulse.
- Start the Time One Pulse and enable interrupt.
- Stop the Time One Pulse and disable interrupt.
- Start the Time One Pulse and enable DMA transfer.
- Stop the Time One Pulse and disable DMA transfer.
- *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()*

39.2.8 Time Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the Time Encoder.
- Stop the Time Encoder.
- Start the Time Encoder and enable interrupt.
- Stop the Time Encoder and disable interrupt.
- Start the Time Encoder and enable DMA transfer.
- Stop the Time Encoder and disable DMA transfer.
- *HAL_TIM_Encoder_Init()*
- *HAL_TIM_Encoder_DeInit()*
- *HAL_TIM_Encoder_MspInit()*
- *HAL_TIM_Encoder_MspDeInit()*
- *HAL_TIM_Encoder_Start()*
- *HAL_TIM_Encoder_Stop()*

- *HAL_TIM_Encoder_Start_IT()*
- *HAL_TIM_Encoder_Stop_IT()*
- *HAL_TIM_Encoder_Start_DMA()*
- *HAL_TIM_Encoder_Stop_DMA()*

39.2.9 IRQ handler management

This section provides Timer IRQ handler function.

- *HAL_TIM_IRQHandler()*

39.2.10 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.
- *HAL_TIM_OC_ConfigChannel()*
- *HAL_TIM_IC_ConfigChannel()*
- *HAL_TIM_PWM_ConfigChannel()*
- *HAL_TIM_OnePulse_ConfigChannel()*
- *HAL_TIM_DMABurst_WriteStart()*
- *HAL_TIM_DMABurst_WriteStop()*
- *HAL_TIM_DMABurst_ReadStart()*
- *HAL_TIM_DMABurst_ReadStop()*
- *HAL_TIM_GenerateEvent()*
- *HAL_TIM_ConfigOCrefClear()*
- *HAL_TIM_ConfigClockSource()*
- *HAL_TIM_ConfigTI1Input()*
- *HAL_TIM_SlaveConfigSynchronization()*
- *HAL_TIM_SlaveConfigSynchronization_IT()*
- *HAL_TIM_ReadCapturedValue()*

39.2.11 TIM Callbacks functions

This section provides TIM callback functions:

- Timer Period elapsed callback
- Timer Output Compare callback
- Timer Input capture callback
- Timer Trigger callback
- Timer Error callback
- *HAL_TIM_PeriodElapsedCallback()*
- *HAL_TIM_OC_DelayElapsedCallback()*
- *HAL_TIM_IC_CaptureCallback()*
- *HAL_TIM_PWM_PulseFinishedCallback()*
- *HAL_TIM_TriggerCallback()*
- *HAL_TIM_ErrorCallback()*

39.2.12 Peripheral State functions

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

- *HAL_TIM_Base_GetState()*
- *HAL_TIM_OC_GetState()*
- *HAL_TIM_PWM_GetState()*
- *HAL_TIM_IC_GetState()*
- *HAL_TIM_OnePulse_GetState()*
- *HAL_TIM_Encoder_GetState()*

39.2.13 HAL_TIM_Base_Init

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• **htim:** : TIM Base handle
Return values	• HAL status

39.2.14 HAL_TIM_Base_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes the TIM Base peripheral.
Parameters	• **htim:** : TIM Base handle
Return values	• HAL status

39.2.15 HAL_TIM_Base_MspInit

Function Name	**void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM Base MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.16 HAL_TIM_Base_MspDeInit

Function Name	**void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes TIM Base MSP.
Parameters	• **htim:** : TIM handle

Return values	• None

39.2.17 HAL_TIM_Base_Start

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)**
Function Description	Starts the TIM Base generation.
Parameters	• **htim:** : TIM handle
Return values	• HAL status

39.2.18 HAL_TIM_Base_Stop

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)**
Function Description	Stops the TIM Base generation.
Parameters	• **htim:** : TIM handle
Return values	• HAL status

39.2.19 HAL_TIM_Base_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)**
Function Description	Starts the TIM Base generation in interrupt mode.
Parameters	• **htim:** : TIM handle
Return values	• HAL status

39.2.20 HAL_TIM_Base_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)**
Function Description	Stops the TIM Base generation in interrupt mode.
Parameters	• **htim:** : TIM handle
Return values	• HAL status

39.2.21 HAL_TIM_Base_Start_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**
Function Description	Starts the TIM Base generation in DMA mode.

Parameters	• **htim:** : TIM handle • **pData:** : The source Buffer address. • **Length:** : The length of data to be transferred from memory to peripheral.
Return values	• HAL status

39.2.22 HAL_TIM_Base_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)**
Function Description	Stops the TIM Base generation in DMA mode.
Parameters	• **htim:** : TIM handle
Return values	• HAL status

39.2.23 HAL_TIM_OC_Init

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• **htim:** : TIM Output Compare handle
Return values	• HAL status

39.2.24 HAL_TIM_OC_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes the TIM peripheral.
Parameters	• **htim:** : TIM Output Compare handle
Return values	• HAL status

39.2.25 HAL_TIM_OC_MspInit

Function Name	**void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM Output Compare MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.26 HAL_TIM_OC_MspDeInit

Function Name	**void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes TIM Output Compare MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.27 HAL_TIM_OC_Start

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Starts the TIM Output Compare signal generation.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.28 HAL_TIM_OC_Stop

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Output Compare signal generation.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.29 HAL_TIM_OC_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Starts the TIM Output Compare signal generation in interrupt mode.
Parameters	• **htim:** : TIM OC handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected

	TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.30 HAL_TIM_OC_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Output Compare signal generation in interrupt mode.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.31 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

39.2.32 HAL_TIM_OC_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Output Compare signal generation in DMA mode.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected

	TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.33 HAL_TIM_PWM_Init

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• **htim:** : TIM handle
Return values	• HAL status

39.2.34 HAL_TIM_PWM_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes the TIM peripheral.
Parameters	• **htim:** : TIM handle
Return values	• HAL status

39.2.35 HAL_TIM_PWM_MspInit

Function Name	**void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM PWM MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.36 HAL_TIM_PWM_MspDeInit

Function Name	**void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes TIM PWM MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.37 HAL_TIM_PWM_Start

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description	Starts the PWM signal generation.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.38 HAL_TIM_PWM_Stop

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the PWM signal generation.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.39 HAL_TIM_PWM_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be 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

39.2.40 HAL_TIM_PWM_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected

	TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.41 HAL_TIM_PWM_Start_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**
Function Description	Starts the TIM PWM signal generation in DMA mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected • **pData:** : The source Buffer address. • **Length:** : The length of data to be transferred from memory to TIM peripheral
Return values	• HAL status

39.2.42 HAL_TIM_PWM_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM PWM signal generation in DMA mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.43 HAL_TIM_IC_Init

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• **htim:** : TIM Input Capture handle
Return values	• HAL status

39.2.44 HAL_TIM_IC_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes the TIM peripheral.
Parameters	• **htim:** : TIM Input Capture handle
Return values	• HAL status

39.2.45 HAL_TIM_IC_MspInit

Function Name	**void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM Input Capture MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.46 HAL_TIM_IC_MspDeInit

Function Name	**void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes TIM Input Capture MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.47 HAL_TIM_IC_Start

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Starts the TIM Input Capture measurement.
Parameters	• **htim:** : TIM Input Capture handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.48 HAL_TIM_IC_Stop

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Input Capture measurement.

Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.49 HAL_TIM_IC_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	• **htim:** : TIM Input Capture handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.50 HAL_TIM_IC_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.51 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 in DMA mode.
Parameters	• **htim:** : TIM Input Capture handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected

	TIM_CHANNEL_4: TIM Channel 4 selected • **pData:** : The destination Buffer address. • **Length:** : The length of data to be transferred from TIM peripheral to memory.
Return values	• HAL status

39.2.52 HAL_TIM_IC_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Input Capture measurement in DMA mode.
Parameters	• **htim:** : TIM Input Capture handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

39.2.53 HAL_TIM_OnePulse_Init

Function Name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)**
Function Description	Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	• **htim:** : TIM OnePulse handle • **OnePulseMode:** : Select the One pulse mode. This parameter can be one of the following values: TIM_OPMODE_SINGLE: Only one pulse will be generated. TIM_OPMODE_REPETITIVE: Repetitive pulses wil be generated.
Return values	• HAL status

39.2.54 HAL_TIM_OnePulse_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes the TIM One Pulse.
Parameters	• **htim:** : TIM One Pulse handle
Return values	• HAL status

39.2.55 HAL_TIM_OnePulse_MspInit

Function Name	**void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)**
Function Description	Initializes the TIM One Pulse MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.56 HAL_TIM_OnePulse_MspDeInit

Function Name	**void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes TIM One Pulse MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.57 HAL_TIM_OnePulse_Start

Function Name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function Description	Starts the TIM One Pulse signal generation.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• HAL status

39.2.58 HAL_TIM_OnePulse_Stop

Function Name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function Description	Stops the TIM One Pulse signal generation.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channels to be disable This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• HAL status

39.2.59 HAL_TIM_OnePulse_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**

Function Description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• HAL status

39.2.60 HAL_TIM_OnePulse_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)**
Function Description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	• **htim:** : TIM One Pulse handle • **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• HAL status

39.2.61 HAL_TIM_Encoder_Init

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)**
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	• **htim:** : TIM Encoder Interface handle • **sConfig:** : TIM Encoder Interface configuration structure
Return values	• HAL status

39.2.62 HAL_TIM_Encoder_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes the TIM Encoder interface.
Parameters	• **htim:** : TIM Encoder handle
Return values	• HAL status

39.2.63 HAL_TIM_Encoder_MspInit

Function Name	**void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)**

Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.64 HAL_TIM_Encoder_MspDeInit

Function Name	**void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes TIM Encoder Interface MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.65 HAL_TIM_Encoder_Start

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Starts the TIM Encoder Interface.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• HAL status

39.2.66 HAL_TIM_Encoder_Stop

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Encoder Interface.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• HAL status

39.2.67 HAL_TIM_Encoder_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description	Starts the TIM Encoder Interface in interrupt mode.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• HAL status

39.2.68 HAL_TIM_Encoder_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Encoder Interface in interrupt mode.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• HAL status

39.2.69 HAL_TIM_Encoder_Start_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)**
Function Description	Starts the TIM Encoder Interface in DMA mode.
Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected • **pData1:** : The destination Buffer address for IC1. • **pData2:** : The destination Buffer address for IC2. • **Length:** : The length of data to be transferred from TIM peripheral to memory.
Return values	• HAL status

39.2.70 HAL_TIM_Encoder_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Encoder Interface in DMA mode.

Parameters	• **htim:** : TIM Encoder Interface handle • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	• HAL status

39.2.71 HAL_TIM_IRQHandler

Function Name	**void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)**
Function Description	This function handles TIM interrupts requests.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.72 HAL_TIM_OC_ConfigChannel

Function Name	**HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)**
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	• **htim:** : TIM Output Compare handle • **sConfig:** : TIM Output Compare configuration structure • **Channel:** : TIM Channels to 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

39.2.73 HAL_TIM_IC_ConfigChannel

Function Name	**HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)**
Function Description	Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.
Parameters	• **htim:** : TIM IC handle • **sConfig:** : TIM Input Capture configuration structure • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected

Return values	• HAL status

39.2.74 HAL_TIM_PWM_ConfigChannel

Function Name	**HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)**
Function Description	Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	• **htim:** : TIM handle • **sConfig:** : TIM PWM 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

39.2.75 HAL_TIM_OnePulse_ConfigChannel

Function Name	**HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)**
Function Description	Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.
Parameters	• **htim:** : TIM One Pulse handle • **sConfig:** : TIM One Pulse configuration structure • **OutputChannel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected • **InputChannel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected
Return values	• HAL status

39.2.76 HAL_TIM_DMABurst_WriteStart

Function Name	**HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)**
Function Description	Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

Parameters	• **htim:** : TIM handle • **BurstBaseAddress:** : TIM Base address from where the DMA will start the Data write This parameter can be one 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 parameter can be one 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

39.2.77 HAL_TIM_DMABurst_WriteStop

Function Name	**HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)**
Function Description	Stops the TIM DMA Burst mode.
Parameters	• **htim:** : TIM handle • **BurstRequestSrc:** : TIM DMA Request sources to disable
Return values	• HAL status

39.2.78 HAL_TIM_DMABurst_ReadStart

Function Name	**HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)**
Function Description	Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.
Parameters	• **htim:** : TIM handle • **BurstBaseAddress:** : TIM Base address from where the DMA will starts the Data read This parameter can be one 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 parameter can be one 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

39.2.79 HAL_TIM_DMABurst_ReadStop

Function Name	**HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)**
Function Description	Stop the DMA burst reading.
Parameters	• **htim:** : TIM handle • **BurstRequestSrc:** : TIM DMA Request sources to disable.
Return values	• HAL status

39.2.80 HAL_TIM_GenerateEvent

Function Name	**HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)**
Function Description	Generate a software event.
Parameters	• **htim:** : TIM handle • **EventSource:** : specifies the event source. This parameter can be one of the following values: TIM_EVENTSOURCE_UPDATE: Timer update Event source TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source TIM_EVENTSOURCE_COM: Timer COM event source TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event

		source TIM_EVENTSOURCE_BREAK: Timer Break event source
Return values	•	HAL status
Notes	•	TIM6 and TIM7 can only generate an update event.
	•	TIM_EVENTSOURCE_COM and TIM_EVENTSOURCE_BREAK are used only with TIM1, TIM15, TIM16 and TIM17.

39.2.81 HAL_TIM_ConfigOCrefClear

Function Name	**HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)**
Function Description	Configures the OCRef clear feature.
Parameters	• **htim:** : TIM handle • **sClearInputConfig:** : pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral. • **Channel:** : specifies the TIM Channel This parameter can be one of the following values: TIM_Channel_1: TIM Channel 1 TIM_Channel_2: TIM Channel 2 TIM_Channel_3: TIM Channel 3 TIM_Channel_4: TIM Channel 4
Return values	• HAL status

39.2.82 HAL_TIM_ConfigClockSource

Function Name	**HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)**
Function Description	Configures the clock source to be used.
Parameters	• **htim:** : TIM handle • **sClockSourceConfig:** : pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.
Return values	• HAL status

39.2.83 HAL_TIM_ConfigTI1Input

Function Name	**HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)**
Function Description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	• **htim:** : TIM handle. • **TI1_Selection:** : Indicate whether or not channel 1 is

	connected to the output of a XOR gate. This parameter can be one of the following values: TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)
Return values	• HAL status

39.2.84 HAL_TIM_SlaveConfigSynchronization

Function Name	**HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**
Function Description	Configures the TIM in Slave mode.
Parameters	• **htim:** : TIM handle. • **sSlaveConfig:** : pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	• HAL status

39.2.85 HAL_TIM_SlaveConfigSynchronization_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**
Function Description	Configures the TIM in Slave mode in interrupt mode.
Parameters	• **htim:** TIM handle. • **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	• HAL status

39.2.86 HAL_TIM_ReadCapturedValue

Function Name	**uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Read the captured value from Capture Compare unit.
Parameters	• **htim:** : TIM handle. • **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1 : TIM

	Channel 1 selected TIM_CHANNEL_2 : TIM Channel 2 selected TIM_CHANNEL_3 : TIM Channel 3 selected TIM_CHANNEL_4 : TIM Channel 4 selected
Return values	• Captured value

39.2.87 HAL_TIM_PeriodElapsedCallback

Function Name	**void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)**
Function Description	Period elapsed callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.88 HAL_TIM_OC_DelayElapsedCallback

Function Name	**void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)**
Function Description	Output Compare callback in non blocking mode.
Parameters	• **htim:** : TIM OC handle
Return values	• None

39.2.89 HAL_TIM_IC_CaptureCallback

Function Name	**void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)**
Function Description	Input Capture callback in non blocking mode.
Parameters	• **htim:** : TIM IC handle
Return values	• None

39.2.90 HAL_TIM_PWM_PulseFinishedCallback

Function Name	**void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)**
Function Description	PWM Pulse finished callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.91 HAL_TIM_TriggerCallback

Function Name	**void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)**

Function Description	Hall Trigger detection callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.92 HAL_TIM_ErrorCallback

Function Name	**void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)**
Function Description	Timer error callback in non blocking mode.
Parameters	• **htim:** : TIM handle
Return values	• None

39.2.93 HAL_TIM_Base_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)**
Function Description	Return the TIM Base state.
Parameters	• **htim:** : TIM Base handle
Return values	• HAL state

39.2.94 HAL_TIM_OC_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)**
Function Description	Return the TIM OC state.
Parameters	• **htim:** : TIM Ouput Compare handle
Return values	• HAL state

39.2.95 HAL_TIM_PWM_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)**
Function Description	Return the TIM PWM state.
Parameters	• **htim:** : TIM handle
Return values	• HAL state

39.2.96 HAL_TIM_IC_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)**

Function Description	Return the TIM Input Capture state.
Parameters	• **htim:** : TIM IC handle
Return values	• HAL state

39.2.97 HAL_TIM_OnePulse_GetState

Function Name	**HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)**
Function Description	Return the TIM One Pulse Mode state.
Parameters	• **htim:** : TIM OPM handle
Return values	• HAL state

39.2.98 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:** : TIM Encoder handle
Return values	• HAL state

39.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

39.3.1 TIM

TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_ENABLE

TIM_AUTOMATICOUTPUT_DISABLE

TIM Break Input Enable Disable

TIM_BREAK_ENABLE

TIM_BREAK_DISABLE

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

TIM_BREAKPOLARITY_HIGH

TIM Channel

TIM_CHANNEL_1

TIM_CHANNEL_2

TIM_CHANNEL_3

TIM_CHANNEL_4

TIM_CHANNEL_ALL

TIM Capture/Compare Channel State

TIM_CCx_ENABLE

TIM_CCx_DISABLE

TIM_CCxN_ENABLE

TIM_CCxN_DISABLE

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 ClearInput Source

TIM_CLEARINPUTSOURCE_ETR

TIM_CLEARINPUTSOURCE_OCREFCLR

TIM_CLEARINPUTSOURCE_NONE

TIM ClockDivision

TIM_CLOCKDIVISION_DIV1

TIM_CLOCKDIVISION_DIV2

TIM_CLOCKDIVISION_DIV4

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_NONINVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_RISING	Polarity for TIx clock sources
TIM_CLOCKPOLARITY_FALLING	Polarity for TIx clock sources
TIM_CLOCKPOLARITY_BOTHEDGE	Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1	No prescaler is used
TIM_CLOCKPRESCALER_DIV2	Prescaler for External ETR Clock: Capture performed once every 2 events.
TIM_CLOCKPRESCALER_DIV4	Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8	Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_ETRMODE2

TIM_CLOCKSOURCE_INTERNAL

TIM_CLOCKSOURCE_ITR0

TIM_CLOCKSOURCE_ITR1

TIM_CLOCKSOURCE_ITR2

TIM_CLOCKSOURCE_ITR3

TIM_CLOCKSOURCE_TI1ED

TIM_CLOCKSOURCE_TI1

TIM_CLOCKSOURCE_TI2

TIM_CLOCKSOURCE_ETRMODE1

TIM Commutation Source

TIM_COMMUTATION_TRGI

TIM_COMMUTATION_SOFTWARE

TIM Counter Mode

TIM_COUNTERMODE_UP

TIM_COUNTERMODE_DOWN

TIM_COUNTERMODE_CENTERALIGNED1

TIM_COUNTERMODE_CENTERALIGNED2

TIM_COUNTERMODE_CENTERALIGNED3

TIM DMA Base Address

TIM_DMABASE_CR1

TIM_DMABASE_CR2

TIM_DMABASE_SMCR

TIM_DMABASE_DIER

TIM_DMABASE_SR

TIM_DMABASE_EGR

TIM_DMABASE_CCMR1

TIM_DMABASE_CCMR2

TIM_DMABASE_CCER

TIM_DMABASE_CNT

TIM_DMABASE_PSC

TIM_DMABASE_ARR

TIM_DMABASE_RCR

TIM_DMABASE_CCR1

TIM_DMABASE_CCR2

TIM_DMABASE_CCR3

TIM_DMABASE_CCR4

TIM_DMABASE_BDTR

TIM_DMABASE_DCR

TIM 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 Handle Index

TIM_DMA_ID_UPDATE	Index of the DMA handle used for Update DMA requests
TIM_DMA_ID_CC1	Index of the DMA handle used for Capture/Compare 1 DMA requests
TIM_DMA_ID_CC2	Index of the DMA handle used for Capture/Compare 2 DMA requests
TIM_DMA_ID_CC3	Index of the DMA handle used for Capture/Compare 3 DMA requests
TIM_DMA_ID_CC4	Index of the DMA handle used for Capture/Compare 4 DMA requests
TIM_DMA_ID_COMMUTATION	Index of the DMA handle used for Commutation DMA requests
TIM_DMA_ID_TRIGGER	Index of the DMA handle used for Trigger DMA requests

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 Event Source

TIM_EVENTSOURCE_UPDATE

TIM_EVENTSOURCE_CC1

TIM_EVENTSOURCE_CC2

TIM_EVENTSOURCE_CC3

TIM_EVENTSOURCE_CC4

TIM_EVENTSOURCE_COM

TIM_EVENTSOURCE_TRIGGER

TIM_EVENTSOURCE_BREAK

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None:

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None:

__HAL_TIM_MOE_ENABLE

Description:

- Enable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None:

__HAL_TIM_DISABLE

Description:

- Disable the TIM peripheral.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None:

__HAL_TIM_MOE_DISABLE

Description:

- Disable the TIM main Output.

Parameters:

- __HANDLE__: TIM handle

Return value:

- None:

__HAL_TIM_ENABLE_IT

Description:

- Enables 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:

- Disables 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:

- Enables 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:

- Disables the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the

following values:
- TIM_DMA_UPDATE: Update DMA request
- TIM_DMA_CC1: Capture/Compare 1 DMA request
- TIM_DMA_CC2: Capture/Compare 2 DMA request
- TIM_DMA_CC3: Capture/Compare 3 DMA request
- TIM_DMA_CC4: Capture/Compare 4 DMA request
- TIM_DMA_COM: Commutation DMA request
- TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None:

__HAL_TIM_GET_FLAG

Description:

- Checks 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_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: 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:

- Clears 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_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: 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:

- Checks whether the specified TIM interrupt has occurred or not.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check.

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.

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)

__HAL_TIM_SET_PRESCALER

Description:

- Sets the TIM active prescaler register value on update event.

Parameters:

- __HANDLE__: TIM handle.
- __PRESC__: specifies the active prescaler register new value.

Return value:

- None:

__HAL_TIM_SET_COMPARE

Description:

- Sets 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
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None:

__HAL_TIM_GET_COMPARE

Description:

- Gets 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

Return value:

- None:

__HAL_TIM_SET_COUNTER

Description:

- Sets 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:

- Gets the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None:

__HAL_TIM_SET_AUTORELOAD

Description:

- Sets 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:

- Gets the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None:

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Sets 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:

- Gets the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None:

__HAL_TIM_SET_ICPRESCALER

Description:

- Sets 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:

- Gets 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:

- 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:

__HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None:

__HAL_TIM_SET_CAPTUREPOLARITY

Description:

- Sets the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
  - TIM_INPUTCHANNELPOLARITY_RISING : Rising Edge
  - TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
  - TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None:

TIM Flag Definition

TIM_FLAG_UPDATE

TIM_FLAG_CC1

TIM_FLAG_CC2

TIM_FLAG_CC3

TIM_FLAG_CC4

TIM_FLAG_COM

TIM_FLAG_TRIGGER

TIM_FLAG_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 level

TIM_LOCKLEVEL_OFF

TIM_LOCKLEVEL_1

TIM_LOCKLEVEL_2

TIM_LOCKLEVEL_3

TIM Master Mode Selection

TIM_TRGO_RESET

TIM_TRGO_ENABLE

TIM_TRGO_UPDATE

TIM_TRGO_OC1

TIM_TRGO_OC1REF

TIM_TRGO_OC2REF

TIM_TRGO_OC3REF

TIM_TRGO_OC4REF

TIM Master Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

TIM_MASTERSLAVEMODE_DISABLE

TIM One Pulse Mode

TIM_OPMODE_SINGLE

TIM_OPMODE_REPETITIVE

TIM OSSI Off State Selection for Idle mode state

TIM_OSSI_ENABLE

TIM_OSSI_DISABLE

TIM OSSR Off State Selection for Run mode state

TIM_OSSR_ENABLE

TIM_OSSR_DISABLE

TIM Output Compare and PWM modes

TIM_OCMODE_TIMING

TIM_OCMODE_ACTIVE

TIM_OCMODE_INACTIVE

TIM_OCMODE_TOGGLE

TIM_OCMODE_PWM1

TIM_OCMODE_PWM2

TIM_OCMODE_FORCED_ACTIVE

TIM_OCMODE_FORCED_INACTIVE

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

TIM_OCIDLESTATE_RESET

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

TIM_OCNIDLESTATE_RESET

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

TIM_OCNPOLARITY_LOW

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

TIM_OUTPUTNSTATE_ENABLE

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

TIM_OUTPUTSTATE_ENABLE

TIM Output Fast State

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

TIM Private Constants

TIM_CCER_CCxE_MASK

TIM_CCER_CCxE_MASK

TIM_CCER_CCxNE_MASK

TIM Private Macros

IS_TIM_COUNTER_MODE

IS_TIM_CLOCKDIVISION_DIV

IS_TIM_PWM_MODE

IS_TIM_OC_MODE

IS_TIM_FAST_STATE

IS_TIM_OC_POLARITY

IS_TIM_OCN_POLARITY

IS_TIM_OCIDLE_STATE

IS_TIM_OCNIDLE_STATE

IS_TIM_CHANNELS

IS_TIM_OPM_CHANNELS

IS_TIM_COMPLEMENTARY_CHANNELS

IS_TIM_IC_POLARITY

IS_TIM_IC_SELECTION

IS_TIM_IC_PRESCALER

IS_TIM_OPM_MODE

IS_TIM_ENCODER_MODE

IS_TIM_DMA_SOURCE

IS_TIM_EVENT_SOURCE

IS_TIM_CLOCKSOURCE

IS_TIM_CLOCKPOLARITY

IS_TIM_CLOCKPRESCALER

IS_TIM_CLOCKFILTER

IS_TIM_CLEARINPUT_SOURCE

IS_TIM_CLEARINPUT_POLARITY

IS_TIM_CLEARINPUT_PRESCALER

IS_TIM_CLEARINPUT_FILTER

IS_TIM_OSSR_STATE

IS_TIM_OSSI_STATE

IS_TIM_LOCK_LEVEL

IS_TIM_BREAK_STATE

IS_TIM_BREAK_POLARITY

IS_TIM_AUTOMATIC_OUTPUT_STATE

IS_TIM_TRGO_SOURCE

IS_TIM_SLAVE_MODE

IS_TIM_MSM_STATE

IS_TIM_TRIGGER_SELECTION

IS_TIM_INTERNAL_TRIGGEREVENT_SELECTION

IS_TIM_TRIGGERPOLARITY

IS_TIM_TRIGGERPRESCALER

IS_TIM_TRIGGERFILTER

IS_TIM_TI1SELECTION

IS_TIM_DMA_BASE

IS_TIM_DMA_LENGTH

IS_TIM_IC_FILTER

TIM_SET_ICPRESCALERVALUE

Description:

- Set TIM IC prescaler.

Parameters:

- __HANDLE__: TIM handle
- __CHANNEL__: specifies TIM Channel
- __ICPSC__: specifies the prescaler value.

Return value:

- None:

TIM_RESET_ICPRESCALERVALUE

Description:

- Reset TIM IC prescaler.

Parameters:

- __HANDLE__: TIM handle
- __CHANNEL__: specifies TIM Channel

Return value:

	• None:
TIM_SET_CAPTUREPOLARITY	**Description:** • Set TIM IC polarity. **Parameters:** • __HANDLE__: TIM handle • __CHANNEL__: specifies TIM Channel • __POLARITY__: specifies TIM Channel Polarity **Return value:** • None:
TIM_RESET_CAPTUREPOLARITY	**Description:** • Reset TIM IC polarity. **Parameters:** • __HANDLE__: TIM handle • __CHANNEL__: specifies TIM Channel **Return value:** • None:

TIM Slave Mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

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

40 HAL TIM Extension Driver

40.1 TIMEx Firmware driver registers structures

40.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the stm32f1xx_hal_tim_ex.h

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

40.1.2 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32f1xx_hal_tim_ex.h

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***
- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode*** TIM off state in run mode This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode*** TIM off state in IDLE mode This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel*** TIM Lock level This parameter can be a value of ***TIM_Lock_level***

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime*** TIM dead Time This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState*** TIM Break State This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity*** TIM Break input polarity This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput*** TIM Automatic Output Enable state This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

40.1.3 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32f1xx_hal_tim_ex.h

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***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::MasterSlaveMode*** Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***

40.2 TIMEx Firmware driver API description

The following section lists the various functions of the TIMEx library.

40.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
   - Output Compare
   - PWM generation (Edge and Center-aligned Mode)
   - One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes

40.2.2 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_TIMEx_HallSensor_MspInit()

2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      – Enable the clock for the TIM GPIOs using the following function: __HAL_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_OCN_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().

40.2.3 Timer Hall Sensor functions

This section provides functions allowing to:

- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.
- ***HAL_TIMEx_HallSensor_Init()***
- ***HAL_TIMEx_HallSensor_DeInit()***
- ***HAL_TIMEx_HallSensor_MspInit()***
- ***HAL_TIMEx_HallSensor_MspDeInit()***
- ***HAL_TIMEx_HallSensor_Start()***
- ***HAL_TIMEx_HallSensor_Stop()***
- ***HAL_TIMEx_HallSensor_Start_IT()***
- ***HAL_TIMEx_HallSensor_Stop_IT()***
- ***HAL_TIMEx_HallSensor_Start_DMA()***
- ***HAL_TIMEx_HallSensor_Stop_DMA()***

40.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.
- ***HAL_TIMEx_OCN_Start()***
- ***HAL_TIMEx_OCN_Stop()***
- ***HAL_TIMEx_OCN_Start_IT()***
- ***HAL_TIMEx_OCN_Stop_IT()***
- ***HAL_TIMEx_OCN_Start_DMA()***
- ***HAL_TIMEx_OCN_Stop_DMA()***

40.2.5 Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.
- ***HAL_TIMEx_PWMN_Start()***
- ***HAL_TIMEx_PWMN_Stop()***
- ***HAL_TIMEx_PWMN_Start_IT()***
- ***HAL_TIMEx_PWMN_Stop_IT()***
- ***HAL_TIMEx_PWMN_Start_DMA()***
- ***HAL_TIMEx_PWMN_Stop_DMA()***

40.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.

- Stop the Complementary One Pulse and disable interrupts.
- *HAL_TIMEx_OnePulseN_Start()*
- *HAL_TIMEx_OnePulseN_Stop()*
- *HAL_TIMEx_OnePulseN_Start_IT()*
- *HAL_TIMEx_OnePulseN_Stop_IT()*

40.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- *HAL_TIMEx_ConfigCommutationEvent()*
- *HAL_TIMEx_ConfigCommutationEvent_IT()*
- *HAL_TIMEx_ConfigCommutationEvent_DMA()*
- *HAL_TIMEx_ConfigBreakDeadTime()*
- *HAL_TIMEx_MasterConfigSynchronization()*

40.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback
- *HAL_TIMEx_CommutationCallback()*
- *HAL_TIMEx_BreakCallback()*
- *TIMEx_DMACommutationCplt()*

40.2.9 Extension Peripheral State functions

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

- *HAL_TIMEx_HallSensor_GetState()*

40.2.10 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 create the associated handle.
Parameters	• **htim:** : TIM Encoder Interface handle • **sConfig:** : TIM Hall Sensor configuration structure
Return values	• HAL status

40.2.11 HAL_TIMEx_HallSensor_DeInit

Function Name	**HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes the TIM Hall Sensor interface.
Parameters	• **htim:** : TIM Hall Sensor handle
Return values	• HAL status

40.2.12 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

40.2.13 HAL_TIMEx_HallSensor_MspDeInit

Function Name	**void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)**
Function Description	DeInitializes TIM Hall Sensor MSP.
Parameters	• **htim:** : TIM handle
Return values	• None

40.2.14 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

40.2.15 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

40.2.16 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

40.2.17 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

40.2.18 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

40.2.19 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

40.2.20 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

40.2.21 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

40.2.22 HAL_TIMEx_OCN_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	• **htim:** : TIM OC handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

40.2.23 HAL_TIMEx_OCN_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be disabled This parameter can

be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

40.2.24 HAL_TIMEx_OCN_Start_DMA

Function Name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**
Function Description	Starts the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected • **pData:** : The source Buffer address. • **Length:** : The length of data to be transferred from memory to TIM peripheral
Return values	• HAL status

40.2.25 HAL_TIMEx_OCN_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	• **htim:** : TIM Output Compare handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

40.2.26 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

40.2.27 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

40.2.28 HAL_TIMEx_PWMN_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Starts the PWM signal generation in interrupt mode on the complementary output.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

40.2.29 HAL_TIMEx_PWMN_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the PWM signal generation in interrupt mode on the complementary output.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected

Return values	• HAL status

40.2.30 HAL_TIMEx_PWMN_Start_DMA

Function Name	**HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)**
Function Description	Starts the TIM PWM signal generation in DMA mode on the complementary output.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected • **pData:** : The source Buffer address. • **Length:** : The length of data to be transferred from memory to TIM peripheral
Return values	• HAL status

40.2.31 HAL_TIMEx_PWMN_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**
Function Description	Stops the TIM PWM signal generation in DMA mode on the complementary output.
Parameters	• **htim:** : TIM handle • **Channel:** : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected TIM_CHANNEL_2: TIM Channel 2 selected TIM_CHANNEL_3: TIM Channel 3 selected TIM_CHANNEL_4: TIM Channel 4 selected
Return values	• HAL status

40.2.32 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 complemetary 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

40.2.33 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

40.2.34 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

40.2.35 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

40.2.36 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.

40.2.37 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.

40.2.38 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

40.2.39 HAL_TIMEx_ConfigBreakDeadTime

Function Name	**HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)**
Function Description	Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).
Parameters	• **htim:** : TIM handle • **sBreakDeadTimeConfig:** : pointer to a

	TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.
Return values	• HAL status

40.2.40 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

40.2.41 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

40.2.42 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

40.2.43 TIMEx_DMACommutationCplt

Function Name	**void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)**
Function Description	TIM DMA Commutation callback.
Parameters	• **hdma:** : pointer to DMA handle.
Return values	• None

40.2.44 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

40.3 TIMEx Firmware driver defines

The following section lists the various define and macros of the module.

40.3.1 TIMEx

TIMEx

TIMEx Clock Filter

IS_TIM_DEADTIME	BreakDead Time

41 HAL UART Generic Driver

41.1 UART Firmware driver registers structures

41.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32f1xx_hal_uart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate*** This member configures the UART communication baud rate. The baud rate is computed using the following formula:
  - IntegerDivider = ((PCLKx) / (16 * (huart->Init.BaudRate)))
  - FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***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_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 wether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***
- ***uint32_t UART_InitTypeDef::HwFlowCtl*** Specifies wether 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 fPCLK/8). This parameter can be a value of ***UART_Over_Sampling***. This feature is not available on STM32F1xx family, so OverSampling parameter should always be set to 16.

41.1.2 UART_HandleTypeDef

UART_HandleTypeDef is defined in the stm32f1xx_hal_uart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***UART_InitTypeDef Init***

- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_UART_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* UART_HandleTypeDef::Instance*** UART registers base address
- ***UART_InitTypeDef UART_HandleTypeDef::Init*** UART communication parameters
- ***uint8_t* UART_HandleTypeDef::pTxBuffPtr*** Pointer to UART Tx transfer Buffer
- ***uint16_t UART_HandleTypeDef::TxXferSize*** UART Tx Transfer size
- ***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
- ***uint16_t UART_HandleTypeDef::RxXferCount*** UART Rx Transfer Counter
- ***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::State*** UART communication state
- ***__IO uint32_t UART_HandleTypeDef::ErrorCode*** UART Error code

41.2 UART Firmware driver API description

The following section lists the various functions of the UART library.

41.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure.
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
   a. Enable the USARTx interface clock.
   b. UART pins configuration:
      - Enable the clock for the UART GPIOs.
      - Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
   c. 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.

d. 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.
   - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)

3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the huart Init structure.
4. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
5. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
6. For the LIN mode, initialize the UART registers by calling the HAL_LIN_Init() API.
7. For the Multi-Processor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.

The specific UART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive process.

These APIs (HAL_UART_Init() and HAL_HalfDuplex_Init()) configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customed HAL_UART_MspInit() API.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_UART_Transmit()
- Receive an amount of data in blocking mode using HAL_UART_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_UART_Transmit_IT()
- At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_UART_Receive_IT()
- At reception end of transfer HAL_UART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxCpltCallback
- In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_UART_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_UART_Transmit_DMA()
- At transmission end of half transfer HAL_UART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxHalfCpltCallback
- At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_UART_Receive_DMA()
- At reception end of half transfer HAL_UART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxHalfCpltCallback
- At reception end of transfer HAL_UART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxCpltCallback
- In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_UART_ErrorCallback
- Pause the DMA Transfer using HAL_UART_DMAPause()
- Resume the DMA Transfer using HAL_UART_DMAResume()
- Stop the DMA Transfer using HAL_UART_DMAStop()

UART HAL driver macros list

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

- __HAL_UART_ENABLE: Enable the UART peripheral
- __HAL_UART_DISABLE: Disable the UART peripheral
- __HAL_UART_GET_FLAG : Check whether the specified UART flag is set or not
- __HAL_UART_CLEAR_FLAG : Clear the specified UART pending flag
- __HAL_UART_ENABLE_IT: Enable the specified UART interrupt
- __HAL_UART_DISABLE_IT: Disable the specified UART interrupt
- __HAL_UART_GET_IT_SOURCE: Check whether the specified UART interrupt has occurred or not

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

41.2.2 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 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. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible UART frame formats are as listed in *Table 22: "UART frame formats"* .

– Hardware flow control
– Receiver/transmitter modes

Table 22: UART frame formats

M bit	PCE bit	UART frame
0	0	| SB | 8 bit data | STB |
0	1	| SB | 7 bit data | PB | STB |
1	0	| SB | 9 bit data | STB |
1	1	| SB | 8 bit data | PB | STB |

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init() and HAL_MultiProcessor_Init() APIs follow respectively the UART asynchronous, UART Half duplex, LIN and Multi-Processor configuration procedures (details for the procedures are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

- ***HAL_UART_Init()***
- ***HAL_HalfDuplex_Init()***
- ***HAL_LIN_Init()***
- ***HAL_MultiProcessor_Init()***
- ***HAL_UART_DeInit()***
- ***HAL_UART_MspInit()***
- ***HAL_UART_MspDeInit()***

41.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the UART asynchronous and Half duplex 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, these APIs return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UART_TxCpltCallback(), HAL_UART_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or receive process. The HAL_UART_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode APIs are:
   – HAL_UART_Transmit()
   – HAL_UART_Receive()
3. Non Blocking mode APIs with Interrupt are:
   – HAL_UART_Transmit_IT()
   – HAL_UART_Receive_IT()
   – HAL_UART_IRQHandler()
4. Non Blocking mode functions with DMA are:
   – HAL_UART_Transmit_DMA()
   – HAL_UART_Receive_DMA()
   – HAL_UART_DMAPause()
   – HAL_UART_DMAResume()

   – HAL_UART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in non blocking mode:
   – HAL_UART_TxHalfCpltCallback()
   – HAL_UART_TxCpltCallback()
   – HAL_UART_RxHalfCpltCallback()
   – HAL_UART_RxCpltCallback()
   – HAL_UART_ErrorCallback()

In the Half duplex communication, it is forbidden to run the transmit and receive process in parallel, the UART state HAL_UART_STATE_BUSY_TX_RX can't be useful.

- ***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_IRQHandler()***
- ***HAL_UART_TxCpltCallback()***
- ***HAL_UART_TxHalfCpltCallback()***
- ***HAL_UART_RxCpltCallback()***
- ***HAL_UART_RxHalfCpltCallback()***
- ***HAL_UART_ErrorCallback()***

41.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART:

- HAL_LIN_SendBreak() API can be helpful to transmit the break character.
- HAL_MultiProcessor_EnterMuteMode() API can be helpful to enter the UART in mute mode.
- HAL_MultiProcessor_ExitMuteMode() API can be helpful to exit the UART mute mode by software.
- HAL_HalfDuplex_EnableTransmitter() API to enable the UART transmitter and disables the UART receiver in Half Duplex mode
- HAL_HalfDuplex_EnableReceiver() API to enable the UART receiver and disables the UART transmitter in Half Duplex mode
- ***HAL_LIN_SendBreak()***
- ***HAL_MultiProcessor_EnterMuteMode()***
- ***HAL_MultiProcessor_ExitMuteMode()***
- ***HAL_HalfDuplex_EnableTransmitter()***
- ***HAL_HalfDuplex_EnableReceiver()***

41.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of UART communication process, return Peripheral Errors occurred during communication process

- HAL_UART_GetState() API can be helpful to check in run-time the state of the UART peripheral.
- HAL_UART_GetError() check in run-time errors that could be occurred during communication.
- ***HAL_UART_GetState()***
- ***HAL_UART_GetError()***

41.2.6 HAL_UART_Init

Function Name	**HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)**
Function Description	Initializes the UART mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

41.2.7 HAL_HalfDuplex_Init

Function Name	**HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)**
Function Description	Initializes the half-duplex mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

41.2.8 HAL_LIN_Init

Function Name	**HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)**
Function Description	Initializes the LIN mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • **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

41.2.9 HAL_MultiProcessor_Init

Function Name	**HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)**
Function Description	Initializes the Multi-Processor mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • **Address:** UART node address • **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

41.2.10 HAL_UART_DeInit

Function Name	**HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)**
Function Description	DeInitializes the UART peripheral.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

41.2.11 HAL_UART_MspInit

Function Name	**void HAL_UART_MspInit (UART_HandleTypeDef * huart)**
Function Description	UART MSP Init.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• None

41.2.12 HAL_UART_MspDeInit

Function Name	**void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)**
Function Description	UART MSP DeInit.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that

contains the configuration information for the specified UART module.

Return values
- None

41.2.13 HAL_UART_Transmit

Function Name	**HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Sends an amount of data in blocking mode.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• HAL status

41.2.14 HAL_UART_Receive

Function Name	**HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Receives an amount of data in blocking mode.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be received • **Timeout:** Timeout duration
Return values	• HAL status

41.2.15 HAL_UART_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function Description	Sends an amount of data in non blocking mode.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• HAL status

41.2.16 HAL_UART_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function Description	Receives an amount of data in non blocking mode.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be received
Return values	• HAL status

41.2.17 HAL_UART_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function Description	Sends an amount of data in non blocking mode.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent
Return values	• HAL status

41.2.18 HAL_UART_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function Description	Receives an amount of data in non blocking mode.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • **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)

41.2.19 HAL_UART_DMAPause

Function Name	**HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)**

Function Description	Pauses the DMA Transfer.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

41.2.20 HAL_UART_DMAResume

Function Name	**HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)**
Function Description	Resumes the DMA Transfer.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

41.2.21 HAL_UART_DMAStop

Function Name	**HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)**
Function Description	Stops the DMA Transfer.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

41.2.22 HAL_UART_IRQHandler

Function Name	**void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)**
Function Description	This function handles UART interrupt request.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• None

41.2.23 HAL_UART_TxCpltCallback

Function Name	**void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)**
Function Description	Tx Transfer completed callbacks.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART

	module.
Return values	• None

41.2.24 HAL_UART_TxHalfCpltCallback

Function Name	**void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)**
Function Description	Tx Half Transfer completed callbacks.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• None

41.2.25 HAL_UART_RxCpltCallback

Function Name	**void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)**
Function Description	Rx Transfer completed callbacks.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• None

41.2.26 HAL_UART_RxHalfCpltCallback

Function Name	**void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)**
Function Description	Rx Half Transfer completed callbacks.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• None

41.2.27 HAL_UART_ErrorCallback

Function Name	**void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)**
Function Description	UART error callbacks.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values	•	None

41.2.28 HAL_LIN_SendBreak

Function Name		**HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)**
Function Description		Transmits break characters.
Parameters	•	**huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	•	HAL status

41.2.29 HAL_MultiProcessor_EnterMuteMode

Function Name		**HAL_StatusTypeDef HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)**
Function Description		Enters the UART in mute mode.
Parameters	•	**huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	•	HAL status

41.2.30 HAL_MultiProcessor_ExitMuteMode

Function Name		**HAL_StatusTypeDef HAL_MultiProcessor_ExitMuteMode (UART_HandleTypeDef * huart)**
Function Description		Exits the UART mute mode: wake up software.
Parameters	•	**huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	•	HAL status

41.2.31 HAL_HalfDuplex_EnableTransmitter

Function Name		**HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)**
Function Description		Enables the UART transmitter and disables the UART receiver.
Parameters	•	**huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	•	HAL status

41.2.32 HAL_HalfDuplex_EnableReceiver

Function Name	**HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)**
Function Description	Enables the UART receiver and disables the UART transmitter.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL status

41.2.33 HAL_UART_GetState

Function Name	**HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)**
Function Description	Returns the UART state.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	• HAL state

41.2.34 HAL_UART_GetError

Function Name	**uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)**
Function Description	Return the UART error code.
Parameters	• **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	• UART Error Code

41.3 UART Firmware driver defines

The following section lists the various define and macros of the module.

41.3.1 UART

UART

UART Error Codes

HAL_UART_ERROR_NONE	No error
HAL_UART_ERROR_PE	Parity error
HAL_UART_ERROR_NE	Noise error
HAL_UART_ERROR_FE	frame error
HAL_UART_ERROR_ORE	Overrun error
HAL_UART_ERROR_DMA	DMA transfer error

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE

Description:

- Reset UART handle state.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_UART_FLUSH_DRREGISTER

Description:

- Flush the UART DR register.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

__HAL_UART_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - UART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5)
  - 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_CLEAR_FLAG

Description:

- Clear the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - 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_RXNE: Receive data register not empty flag.

Return value:

- None:

__HAL_UART_CLEAR_PEFLAG

Description:

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_UART_CLEAR_FEFLAG

Description:

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART

Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_UART_CLEAR_NEFLAG

Description:

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_UART_CLEAR_OREFLAG

Description:

- Clear the UART ORE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_UART_CLEAR_IDLEFLAG

Description:

- Clear the UART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral

(USART,UART availability and x,y values depending on device).

- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  – 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. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  – 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_SOURCE

Description:

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- __IT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - 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_ERR: Error interrupt

Return value:

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

__HAL_UART_HWCONTROL_CTS_ENABLE

Description:

- Enable CTS flow control This macro allows to enable CTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control This macro allows to disable CTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control This macro allows to enable RTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control This macro allows to disable RTS hardware flow control for a given UART instance, without need to call

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_UART_ENABLE

Description:

- Enable UART.

Parameters:

- __HANDLE__: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

__HAL_UART_DISABLE

Description:

- Disable UART UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None:

UART FLags

UART_FLAG_CTS

UART_FLAG_LBD

UART_FLAG_TXE

UART_FLAG_TC

UART_FLAG_RXNE

UART_FLAG_IDLE

UART_FLAG_ORE

UART_FLAG_NE

UART_FLAG_FE

UART_FLAG_PE

UART Hardware Flow Control

UART_HWCONTROL_NONE

UART_HWCONTROL_RTS

UART_HWCONTROL_CTS

UART_HWCONTROL_RTS_CTS

UART Interrupt Definitions

UART_IT_PE

UART_IT_TXE

UART_IT_TC

UART_IT_RXNE

UART_IT_IDLE

UART_IT_LBD

UART_IT_CTS

UART_IT_ERR

UART LIN Break Detection Length

UART_LINBREAKDETECTLENGTH_10B

UART_LINBREAKDETECTLENGTH_11B

UART Transfer Mode

UART_MODE_RX

UART_MODE_TX

UART_MODE_TX_RX

UART Over Sampling

UART_OVERSAMPLING_16

UART Parity

UART_PARITY_NONE

UART_PARITY_EVEN

UART_PARITY_ODD

UART Private Macros

UART_CR1_REG_INDEX

UART_CR2_REG_INDEX

UART_CR3_REG_INDEX

UART_DIV_SAMPLING16

UART_DIVMANT_SAMPLING16

UART_DIVFRAQ_SAMPLING16

UART_BRR_SAMPLING16

IS_UART_WORD_LENGTH

IS_UART_LIN_WORD_LENGTH

IS_UART_STOPBITS

IS_UART_PARITY

IS_UART_HARDWARE_FLOW_CONTROL

IS_UART_MODE

IS_UART_STATE

IS_UART_OVERSAMPLING

IS_UART_LIN_OVERSAMPLING

IS_UART_LIN_BREAK_DETECT_LENGTH

IS_UART_WAKEUPMETHOD	
IS_UART_BAUDRATE	72 MHz) divided by the smallest oversampling used on the USART (i.e. 16) Retrun : TRUE or FALSE
IS_UART_ADDRESS	This parameter must be a number between Min_Data = 0 and Max_Data = 15 Return : TRUE or FALSE
UART_IT_MASK	

UART State

UART_STATE_DISABLE

UART_STATE_ENABLE

UART Number of Stop Bits

UART_STOPBITS_1

UART_STOPBITS_2

UART Wakeup Functions

UART_WAKEUPMETHOD_IDLELINE

UART_WAKEUPMETHOD_ADDRESSMARK

UART Word Length

UART_WORDLENGTH_8B

UART_WORDLENGTH_9B

42 HAL USART Generic Driver

42.1 USART Firmware driver registers structures

42.1.1 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32f1xx_hal_usart.h

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***

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate*** This member configures the Usart communication baud rate. The baud rate is computed using the following formula:
  - IntegerDivider = ((PCLKx) / (16 * (husart->Init.BaudRate)))
  - FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5
- ***uint32_t USART_InitTypeDef::WordLength*** Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_Word_Length***
- ***uint32_t USART_InitTypeDef::StopBits*** Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***
- ***uint32_t USART_InitTypeDef::Parity*** Specifies the parity mode. This parameter can be a value of ***USART_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t USART_InitTypeDef::Mode*** Specifies wether 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***

42.1.2 USART_HandleTypeDef

USART_HandleTypeDef is defined in the stm32f1xx_hal_usart.h

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***
- ***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
- ***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

42.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

42.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure.
2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
   a. Enable the USARTx interface clock.
   b. 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).

c. 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.

d. 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 initilalized 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.
   - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)

3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the husart Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
   - These APIs configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customed HAL_USART_MspInit(&husart) API. 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.
5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_USART_Transmit()
- Receive an amount of data in blocking mode using HAL_USART_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_USART_Transmit_IT()
- At transmission end of transfer HAL_USART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_USART_Receive_IT()
- At reception end of transfer HAL_USART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxCpltCallback
- In case of transfer Error, HAL_USART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_USART_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_USART_Transmit_DMA()
- At transmission end of half transfer HAL_USART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxHalfCpltCallback
- At transmission end of transfer HAL_USART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_USART_Receive_DMA()
- At reception end of half transfer HAL_USART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxHalfCpltCallback
- At reception end of transfer HAL_USART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxCpltCallback
- In case of transfer Error, HAL_USART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_USART_ErrorCallback
- Pause the DMA Transfer using HAL_USART_DMAPause()
- Resume the DMA Transfer using HAL_USART_DMAResume()
- Stop the DMA Transfer using HAL_USART_DMAStop()

USART HAL driver macros list

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

- __HAL_USART_ENABLE: Enable the USART peripheral
- __HAL_USART_DISABLE: Disable the USART peripheral
- __HAL_USART_GET_FLAG : Check whether the specified USART flag is set or not
- __HAL_USART_CLEAR_FLAG : Clear the specified USART pending flag
- __HAL_USART_ENABLE_IT: Enable the specified USART interrupt
- __HAL_USART_DISABLE_IT: Disable the specified USART interrupt
- __HAL_USART_GET_IT_SOURCE: Check whether the specified USART interrupt has occurred or not

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

42.2.2 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. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible USART frame formats are as listed in *Table 23: "USART frame formats"*.
  - USART polarity

- USART phase
- USART LastBit
- Receiver/transmitter modes

Table 23: USART frame formats

M bit	PCE bit	UART frame
0	0	| SB | 8 bit data | STB |
0	1	| SB | 7 bit data | PB | STB |
1	0	| SB | 9 bit data | STB |
1	1	| SB | 8 bit data | PB | STB |

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manuals (RM0008 for STM32F10Xxx MCUs and RM0041 for STM32F100xx MCUs)).

- ***HAL_USART_Init()***
- ***HAL_USART_DeInit()***
- ***HAL_USART_MspInit()***
- ***HAL_USART_MspDeInit()***

42.2.3 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 APIs are :
   - HAL_USART_Transmit() in simplex mode
   - HAL_USART_Receive() in full duplex receive only
   - HAL_USART_TransmitReceive() in full duplex mode
3. Non Blocking mode APIs with Interrupt are :
   - HAL_USART_Transmit_IT()in simplex mode
   - HAL_USART_Receive_IT() in full duplex receive only
   - HAL_USART_TransmitReceive_IT() in full duplex mode
   - HAL_USART_IRQHandler()
4. Non Blocking mode functions 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_TxHalfCpltCallback()
  - HAL_USART_TxCpltCallback()
  - HAL_USART_RxHalfCpltCallback()
  - HAL_USART_RxCpltCallback()
  - HAL_USART_ErrorCallback()
  - HAL_USART_TxRxCpltCallback()

- ***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_IRQHandler()***
- ***HAL_USART_TxCpltCallback()***
- ***HAL_USART_TxHalfCpltCallback()***
- ***HAL_USART_RxCpltCallback()***
- ***HAL_USART_RxHalfCpltCallback()***
- ***HAL_USART_TxRxCpltCallback()***
- ***HAL_USART_ErrorCallback()***

42.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of USART communication process, return Peripheral Errors occurred during communication process

- HAL_USART_GetState() API can be helpful to check in run-time the state of the USART peripheral.
- HAL_USART_GetError() check in run-time errors that could be occurred during communication.
- ***HAL_USART_GetState()***
- ***HAL_USART_GetError()***

42.2.5 HAL_USART_Init

Function Name	**HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)**
Function Description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create the associated handle.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified

		USART module.
Return values	•	HAL status

42.2.6 HAL_USART_DeInit

Function Name	**HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)**
Function Description	DeInitializes the USART peripheral.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• HAL status

42.2.7 HAL_USART_MspInit

Function Name	**void HAL_USART_MspInit (USART_HandleTypeDef * husart)**
Function Description	USART MSP Init.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.8 HAL_USART_MspDeInit

Function Name	**void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)**
Function Description	USART MSP DeInit.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.9 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:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pTxData:** Pointer to data buffer

- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values
- HAL status

42.2.10 HAL_USART_Receive

Function Name	**HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**
Function Description	Full-Duplex Receive an amount of data in blocking mode.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pRxData:** Pointer to data buffer • **Size:** Amount of data to be received • **Timeout:** Timeout duration
Return values	• HAL status

42.2.11 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 receive an amount of data in full-duplex mode (blocking mode).
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pTxData:** Pointer to data transmitted buffer • **pRxData:** Pointer to data received buffer • **Size:** Amount of data to be sent • **Timeout:** Timeout duration
Return values	• HAL status

42.2.12 HAL_USART_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)**
Function Description	Simplex Send an amount of data in non-blocking mode.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pTxData:** Pointer to data buffer

		• **Size:** Amount of data to be sent
Return values		• HAL status
Notes		• The USART errors are not managed to avoid the overrun error.

42.2.13 HAL_USART_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)**
Function Description	Simplex Receive an amount of data in non-blocking mode.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pRxData:** Pointer to data buffer • **Size:** Amount of data to be received
Return values	• HAL status

42.2.14 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 receive an amount of data in full-duplex mode (non-blocking).
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pTxData:** Pointer to data transmitted buffer • **pRxData:** Pointer to data received buffer • **Size:** Amount of data to be received
Return values	• HAL status

42.2.15 HAL_USART_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)**
Function Description	Simplex Send an amount of data in non-blocking mode.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pTxData:** Pointer to data buffer • **Size:** Amount of data to be sent

Return values	• HAL status

42.2.16 HAL_USART_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)**
Function Description	Full-Duplex Receive an amount of data in non-blocking mode.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pRxData:** Pointer to data buffer • **Size:** Amount of data to be received
Return values	• HAL status
Notes	• The USART DMA transmit channel must be configured in order to generate the clock for the slave. • When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

42.2.17 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:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • **pTxData:** Pointer to data transmitted buffer • **pRxData:** Pointer to data received buffer • **Size:** Amount of data to be received
Return values	• HAL status
Notes	• When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

42.2.18 HAL_USART_DMAPause

Function Name	**HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)**
Function Description	Pauses the DMA Transfer.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values	• HAL status

42.2.19 HAL_USART_DMAResume

Function Name	**HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)**
Function Description	Resumes the DMA Transfer.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• HAL status

42.2.20 HAL_USART_DMAStop

Function Name	**HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)**
Function Description	Stops the DMA Transfer.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• HAL status

42.2.21 HAL_USART_IRQHandler

Function Name	**void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)**
Function Description	This function handles USART interrupt request.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.22 HAL_USART_TxCpltCallback

Function Name	**void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)**
Function Description	Tx Transfer completed callbacks.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.23 HAL_USART_TxHalfCpltCallback

Function Name	**void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)**
Function Description	Tx Half Transfer completed callbacks.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.24 HAL_USART_RxCpltCallback

Function Name	**void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)**
Function Description	Rx Transfer completed callbacks.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.25 HAL_USART_RxHalfCpltCallback

Function Name	**void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)**
Function Description	Rx Half Transfer completed callbacks.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.26 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:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.27 HAL_USART_ErrorCallback

Function Name	**void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)**
Function Description	USART error callbacks.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• None

42.2.28 HAL_USART_GetState

Function Name	**HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)**
Function Description	Returns the USART state.
Parameters	• **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	• HAL state

42.2.29 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 Error Code

42.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

42.3.1 USART

USART

USART Clock

USART_CLOCK_DISABLE

USART_CLOCK_ENABLE

USART Clock Phase

USART_PHASE_1EDGE

USART_PHASE_2EDGE

USART Clock Polarity

USART_POLARITY_LOW

USART_POLARITY_HIGH

USART Error Codes

HAL_USART_ERROR_NONE	No error
HAL_USART_ERROR_PE	Parity error
HAL_USART_ERROR_NE	Noise error
HAL_USART_ERROR_FE	frame error
HAL_USART_ERROR_ORE	Overrun error
HAL_USART_ERROR_DMA	DMA transfer error

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_USART_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - 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_CLEAR_FLAG

Description:

- Clear the specified USART pending flags.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - USART_FLAG_TC: Transmission Complete flag.
  - USART_FLAG_RXNE: Receive data register not empty flag.

Return value:

- None:

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_USART_ENABLE_IT

Description:

- Enable the specified Usart interrupts.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - 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 interrupts.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - 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_SOURCE

Description:

- Check whether the specified Usart interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - 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_ERR: Error interrupt
- USART_IT_PE: Parity Error interrupt

Return value:

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

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None:

USART Flags

USART_FLAG_CTS

USART_FLAG_LBD

USART_FLAG_TXE

USART_FLAG_TC

USART_FLAG_RXNE

USART_FLAG_IDLE

USART_FLAG_ORE

USART_FLAG_NE

USART_FLAG_FE

USART_FLAG_PE

USART Interrupts Definition

USART_IT_PE

USART_IT_TXE

USART_IT_TC

USART_IT_RXNE

USART_IT_IDLE

USART_IT_LBD

USART_IT_CTS

USART_IT_ERR

USART Last Bit

USART_LASTBIT_DISABLE

USART_LASTBIT_ENABLE

USART Mode

USART_MODE_RX

USART_MODE_TX

USART_MODE_TX_RX

USART NACK State

USART_NACK_ENABLE

USART_NACK_DISABLE

USART Parity

USART_PARITY_NONE

USART_PARITY_EVEN

USART_PARITY_ODD

USART Private Constants

DUMMY_DATA

USART Private Macros

USART_CR1_REG_INDEX

USART_CR2_REG_INDEX

USART_CR3_REG_INDEX

USART_DIV

USART_DIVMANT

USART_DIVFRAQ

USART_BRR

IS_USART_BAUDRATE	72 MHz) divided by the smallest oversampling used on the USART (i.e. 16) return : TRUE or FALSE

IS_USART_WORD_LENGTH

IS_USART_STOPBITS

IS_USART_PARITY

IS_USART_MODE

IS_USART_CLOCK

IS_USART_POLARITY

IS_USART_PHASE

IS_USART_LASTBIT

IS_USART_NACK_STATE

USART_IT_MASK

USART Number of Stop Bits

USART_STOPBITS_1

USART_STOPBITS_0_5

USART_STOPBITS_2

USART_STOPBITS_1_5

USART Word Length

USART_WORDLENGTH_8B

USART_WORDLENGTH_9B

43 HAL WWDG Generic Driver

43.1 WWDG Firmware driver registers structures

43.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32f1xx_hal_wwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***

Field Documentation

- ***uint32_t WWDG_InitTypeDef::Prescaler*** Specifies the prescaler value of the WWDG. This parameter can be a value of ***WWDG_Prescaler***
- ***uint32_t WWDG_InitTypeDef::Window*** Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number lower than Max_Data = 0x80
- ***uint32_t WWDG_InitTypeDef::Counter*** Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F

43.1.2 WWDG_HandleTypeDef

WWDG_HandleTypeDef is defined in the stm32f1xx_hal_wwdg.h

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_WWDG_StateTypeDef State***

Field Documentation

- ***WWDG_TypeDef* WWDG_HandleTypeDef::Instance*** Register base address
- ***WWDG_InitTypeDef WWDG_HandleTypeDef::Init*** WWDG required parameters
- ***HAL_LockTypeDef WWDG_HandleTypeDef::Lock*** WWDG locking object
- ***__IO HAL_WWDG_StateTypeDef WWDG_HandleTypeDef::State*** WWDG communication state

43.2 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

43.2.1 WWDG specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the Counter (T[6;0] downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWDG cannot be disabled except by a system reset.
- WWDGRST flag in RCC_CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- WWDG clock (Hz) = PCLK / (4096 * Prescaler)
- WWDG timeout (mS) = 1000 * (T[5;0] + 1) / WWDG clock where T[5;0] are the lowest 6 bits of Counter.
- WWDG Counter refresh is allowed between the following limits :
  - min time (mS) = 1000 * (Counter - Window) / WWDG clock
  - max time (mS) = 1000 * (Counter - 0x40) / WWDG clock
- Min-max timeout value @48 MHz(PCLK): ~85,3us / ~5,46 ms

43.2.2 How to use this driver

- Enable WWDG APB1 clock using __HAL_RCC_WWDG_CLK_ENABLE().
- Set the WWDG prescaler, refresh window and counter value using HAL_WWDG_Init() function.
- Start the WWDG using HAL_WWDG_Start() function. When the WWDG is enabled the counter value should be configured to a value greater than 0x40 to prevent generating an immediate reset.
- Optionally you can enable the Early Wakeup Interrupt (EWI) which is generated when the counter reaches 0x40, and then start the WWDG using HAL_WWDG_Start_IT(). At EWI HAL_WWDG_WakeupCallback is executed and user can add his own code by customization of function pointer HAL_WWDG_WakeupCallback Once enabled, EWI interrupt cannot be disabled except by a system reset.
- Then 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 refresh window value already programmed.

WWDG HAL driver macros list

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

- __HAL_WWDG_ENABLE: Enable the WWDG peripheral
- __HAL_WWDG_GET_FLAG: Get the selected WWDG's flag status
- __HAL_WWDG_CLEAR_FLAG: Clear the WWDG's pending flags
- __HAL_WWDG_ENABLE_IT: Enables the WWDG early wakeup interrupt

43.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the WWDG according to the specified parameters in the WWDG_InitTypeDef and create the associated handle
- DeInitialize the WWDG peripheral
- Initialize the WWDG MSP
- DeInitialize the WWDG MSP
- ***HAL_WWDG_Init()***
- ***HAL_WWDG_DeInit()***
- ***HAL_WWDG_MspInit()***
- ***HAL_WWDG_MspDeInit()***
- ***HAL_WWDG_WakeupCallback()***

43.2.4 IO operation functions

This section provides functions allowing to:

- Start the WWDG.
- Refresh the WWDG.
- Handle WWDG interrupt request.
- ***HAL_WWDG_Start()***
- ***HAL_WWDG_Start_IT()***
- ***HAL_WWDG_Refresh()***
- ***HAL_WWDG_IRQHandler()***
- ***HAL_WWDG_WakeupCallback()***

43.2.5 Peripheral State functions

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

- ***HAL_WWDG_GetState()***

43.2.6 HAL_WWDG_Init

Function Name	**HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)**
Function Description	Initializes the WWDG according to the specified parameters in the WWDG_InitTypeDef and creates the associated handle.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• HAL status

43.2.7 HAL_WWDG_DeInit

Function Name	**HAL_StatusTypeDef HAL_WWDG_DeInit (WWDG_HandleTypeDef * hwwdg)**
Function Description	DeInitializes the WWDG peripheral.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified

	WWDG module.
Return values	• HAL status

43.2.8 HAL_WWDG_MspInit

Function Name	**void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)**
Function Description	Initializes the WWDG MSP.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• None

43.2.9 HAL_WWDG_MspDeInit

Function Name	**void HAL_WWDG_MspDeInit (WWDG_HandleTypeDef * hwwdg)**
Function Description	DeInitializes the WWDG MSP.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• None

43.2.10 HAL_WWDG_WakeupCallback

Function Name	**void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwwdg)**
Function Description	Early Wakeup WWDG callback.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• None

43.2.11 HAL_WWDG_Start

Function Name	**HAL_StatusTypeDef HAL_WWDG_Start (WWDG_HandleTypeDef * hwwdg)**
Function Description	Starts the WWDG.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• HAL status

43.2.12 HAL_WWDG_Start_IT

Function Name	**HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDef * hwwdg)**
Function Description	Starts the WWDG with interrupt enabled.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• HAL status

43.2.13 HAL_WWDG_Refresh

Function Name	**HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg, uint32_t Counter)**
Function Description	Refreshes the WWDG.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module. • **Counter:** value of counter to put in WWDG counter
Return values	• HAL status

43.2.14 HAL_WWDG_IRQHandler

Function Name	**void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)**
Function Description	Handles 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 when calling HAL_WWDG_Start_IT function. 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.

43.2.15 HAL_WWDG_WakeupCallback

Function Name	**void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwwdg)**

Function Description	Early Wakeup WWDG callback.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• None

43.2.16 HAL_WWDG_GetState

Function Name	**HAL_WWDG_StateTypeDef HAL_WWDG_GetState (WWDG_HandleTypeDef * hwwdg)**
Function Description	Returns the WWDG state.
Parameters	• **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	• HAL state

43.3 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

43.3.1 WWDG

WWDG

WWDG Exported Macros

__HAL_WWDG_RESET_HANDLE_STATE

Description:

- Reset WWDG handle state.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None:

__HAL_WWDG_ENABLE

Description:

- Enables the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None:

__HAL_WWDG_DISABLE

Description:

- Disables the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None:

__HAL_WWDG_ENABLE_IT

Description:

- Enables 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:

__HAL_WWDG_DISABLE_IT

Description:

- Disables the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to disable. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None:

__HAL_WWDG_GET_IT

Description:

- Gets the selected WWDG's it status.

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's interrupt pending bits bits to clear the selected 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:

- Gets the selected WWDG's flag status.

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:

- Clears 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:

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

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 Private Macros

IS_WWDG_PRESCALER

IS_WWDG_WINDOW

IS_WWDG_COUNTER

44 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not required 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 STM32F1 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32F1 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 4.4: "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: stm32f1xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration…)

A template is provided in the HAL drivers folders (stm32f1xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32f1xx_hal.h file has to be included.

What is the difference between stm32f1xx_hal_ppp.c/.h and stm32f1xx_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 (stm32f1xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32f1xx_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_stm32f1xx.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 stm32f1xx_hal_msp.c. A template is provided in the HAL driver folders (stm32f1xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute __*weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,…), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The sysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in Systick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).
2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() in stm32f1xx_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 stm32f1xx_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 stm32f1xx_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.

45 Revision history

Table 24: Document revision history

Date	Revision	Changes
05-Feb-2015	1	Initial release.