UM1786
User manual

Description of STM32F3xx HAL drivers

Introduction

STMCube[TM] 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 STM32CubeF3 for STM32F3 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[TM] 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
COMP	Comparator
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRYP	Cryptographic processor unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
EXTI	External interrupt/event controller
FLASH	Flash memory
FMC	Flexible Memory Controller
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HRTIM	High Resolution Timer
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
RNG	Random Number Generator
RTC	Real-time clock
SD	Secure Digital
SDADC	Sigma-delta Analog-to-digital Converter
SRAM	SRAM external memory

Acronym	Definition
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch Sensing Controller
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

2 Overview of HAL drivers

The HAL drivers 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
stm32f3xx_hal_ppp.c	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. *Example: stm32f3xx_hal_adc.c, stm32f3xx_hal_irda.c, …*
stm32f3xx_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: stm32f3xx_hal_adc.h, stm32f3xx_hal_irda.h, …*

File	Description
stm32f3xx_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: stm32f3xx_hal_adc_ex.c, stm32f3xx_hal_dma_ex.c, …*
stm32f3xx_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: stm32f3xx_hal_adc_ex.h, stm32f3xx_hal_dma_ex.h, …*
stm32f3xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
stm32f3xx_hal.h	stm32f3xx_hal.c header file
stm32f3xx_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.
stm32f3xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application.
stm32f3xx_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_stm32f3xx.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_stm32f3xx.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.
stm32f3xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
stm32f3xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32f3xx_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
stm32f3xx_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 *stm32f3xx*_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.

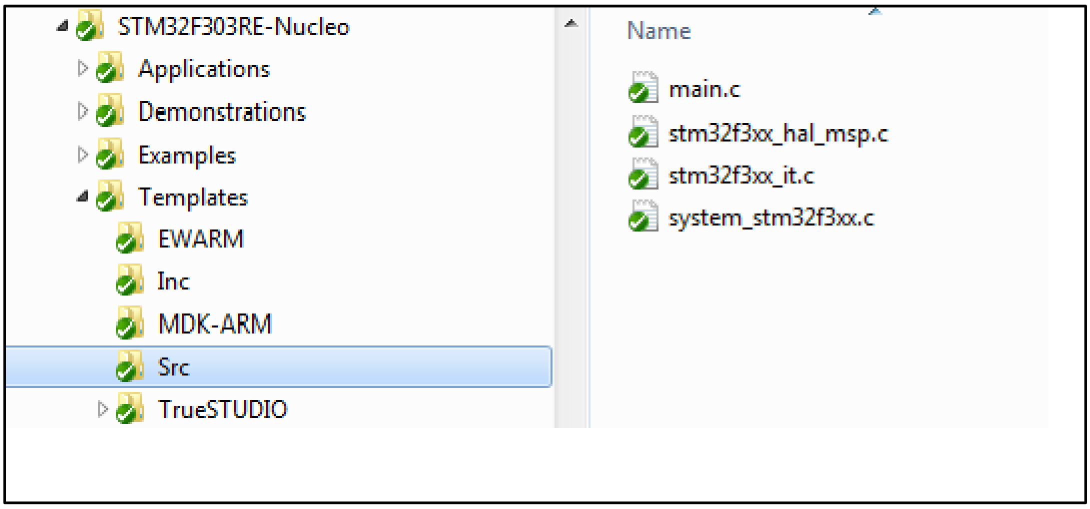
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

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(STM32F302xC) || defined(STM32F303xC) || defined(STM32F358xx) || \
defined(STM32F303x8) || defined(STM32F334x8) || defined(STM32F328xx) || \
defined(STM32F301x8) || defined(STM32F302x8) || defined(STM32F318xx) || \
defined(STM32F373xC) || defined(STM32F378xx)
#endif /* STM32F302xC || STM32F303xC || STM32F358xx || */
/* STM32F303x8 || STM32F334x8 || STM32F328xx || */
/* STM32F301x8 || STM32F302x8 || STM32F318xx */
/* STM32F373xC || STM32F378xx */
```

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

IP/Module	STM32F301x6/x8	STM32F302x6/x8	STM32F302xB/xC	STM32F302xE	STM32F303x6/x8	STM32F303xC	STM32F303xE	STM32F373xB/xC	STM32F334x6/xSTM32F373xB/C8	STM32F318xx	STM32F328xx	STM32F358xx	STM32F378xx	STM32F398xx
stm32f3xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_can.c	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
stm32f3xx_hal_cec.c	No	No	No	No	No	No	No	Yes	No	No	No	No	Yes	No
stm32f3xx_hal_comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_dac_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_flash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_hrtim.c	No	No	No	No	No	No	No	No	Yes	No	Yes	No	No	No
stm32f3xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32F301x6/x8	STM32F302x6/x8	STM32F302xB/xC	STM32F302xE	STM32F303x6/x8	STM32F303xC	STM32F303xE	STM32F373xB/xC	STM32F334x6/xSTM32F373xB/C8	STM32F318xx	STM32F328xx	STM32F358xx	STM32F378xx	STM32F398xx
stm32f3xx_hal_i2s.c	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32f3xx_hal_i2s_ex.c	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
stm32f3xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_msp_template.c	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stm32f3xx_hal_nand.c	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	Yes
stm32f3xx_hal_nor.c	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	Yes
stm32f3xx_hal_opamp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes
stm32f3xx_hal_opamp_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes
stm32f3xx_hal_pcd.c	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No	No	Yes
stm32f3xx_hal_pcd_ex.c	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No	No	Yes
stm32f3xx_hal_pccard.c	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	Yes
stm32f3xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes
stm32f3xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

IP/Module	STM32F301x6/x8	STM32F302x6/x8	STM32F302xB/xC	STM32F302xE	STM32F303x6/x8	STM32F303xC	STM32F303xE	STM32F373xB/xC	STM32F334x6/xST M32F373xB/C8	STM32F318xx	STM32F328xx	STM32F358xx	STM32F378xx	STM32F398xx
stm32f3xx_hal_sdadc.c	No	No	No	No	No	No	No	Yes	No	No	No	No	Yes	No
stm32f3xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_smartcard_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_sram.c	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	Yes
stm32f3xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_tsc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32f3xx_ll_fmc.c	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	Yes

2.5 HAL drivers rules

2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6: HAL API naming rules

	Generic	Family specific	Device specific
File names	*stm32f3xx_hal_ppp (c/h)*	*stm32f3xx_hal_ppp_ex (c/h)*	*stm32f3xx_ 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 STM32F3xx reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (e.g. ADC_TypeDef) in the CMSIS header file corresponding to the selected platform: stm32f301x8.h, stm32f302x8.h, stm32f302xc.h, stm32f302xe.h, stm32f303x8.h, stm32f303xc.h, stm32f303xe.h, stm32f318xx.h, stm32f328xx.h, stm32f334x8.h, stm32f358xx.h, stm32f373xc.h, stm32f378xx.h and stm32f398xx.h . The platform is selected by enabling the compilation switch in the compilation toolchain directive or in the stm32f3xx.h file.
- 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 7: 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 stm32f3xx_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 stm32f3xx_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 8: 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 9: 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, stm32f3xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32f3xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 10: HAL extension APIs

Function Group	Common API Name
HAL_ADCEx_Calibration_Start()	This function is used to start the automatic ADC calibration
HAL_ADCEx_Calibration_GetValue()	This function is used to get the ADC calibration factor
HAL_ADCEx_Calibration_SetValue()	This function is used to set the calibration factor to overwrite automatic conversion result

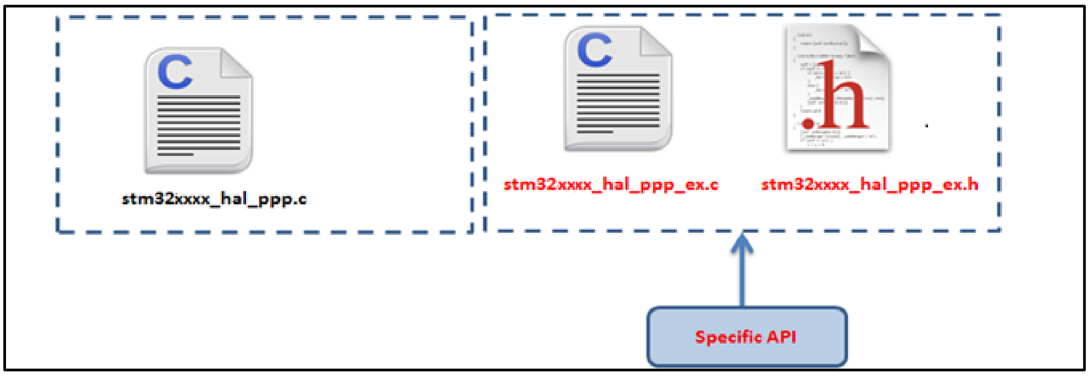
2.7.2 HAL extension model cases

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 stm32f3xx_hal_adc_ex.c extension file. They are named HAL_PPPEx_Function().

Figure 2: Adding device-specific functions



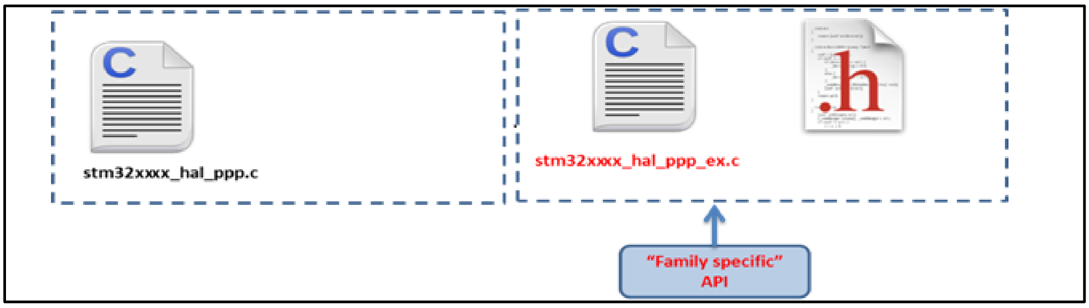
Example: stm32f3xx_hal_adc_ex.c/h

```
#if defined(STM32F302xE) || defined(STM32F303xE) || defined(STM32F398xx) || \
defined(STM32F302xC) || defined(STM32F303xC) || defined(STM32F358xx) || \
defined(STM32F303x8) || defined(STM32F334x8) || defined(STM32F328xx) || \
defined(STM32F301x8) || defined(STM32F302x8) || defined(STM32F318xx)
HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(struct __ADC_HandleTypeDef* hadc,
uint32_t SingleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(struct __ADC_HandleTypeDef *hadc, uint32_t
SingleDiff);
HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue(struct __ADC_HandleTypeDef *hadc,
uint32_t SingleDiff, uint32_t CalibrationFactor);
#endif /* STM32F302xE || STM32F303xE || STM32F398xx || */
/* STM32F302xC || STM32F303xC || STM32F358xx || */
/* STM32F303x8 || STM32F334x8 || STM32F328xx || */
/* STM32F301x8 || STM32F302x8 || STM32F318xx */
```

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

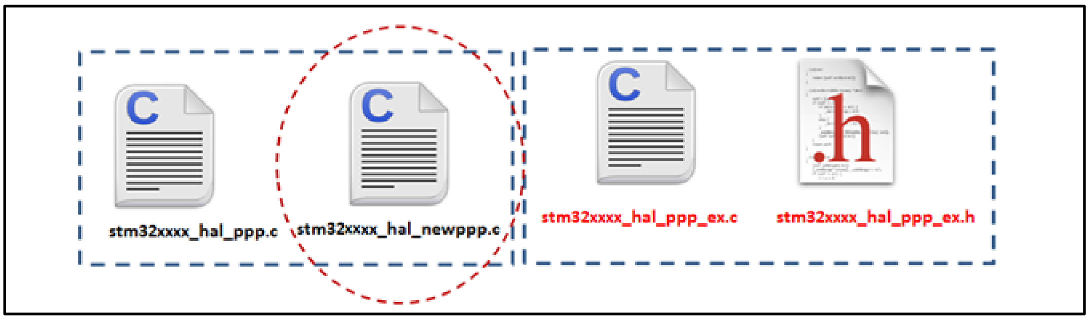


Case3 : Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in stm32f3xx_hal_newppp.c. However the inclusion of this file is selected in the stm32f3xx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

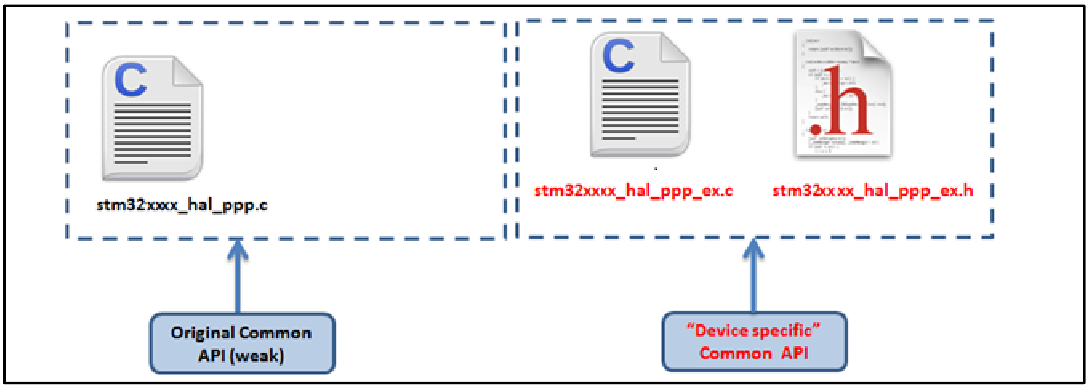


Example: stm32f3xx_hal_sdadc.c/h

Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the stm32f3xx_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.

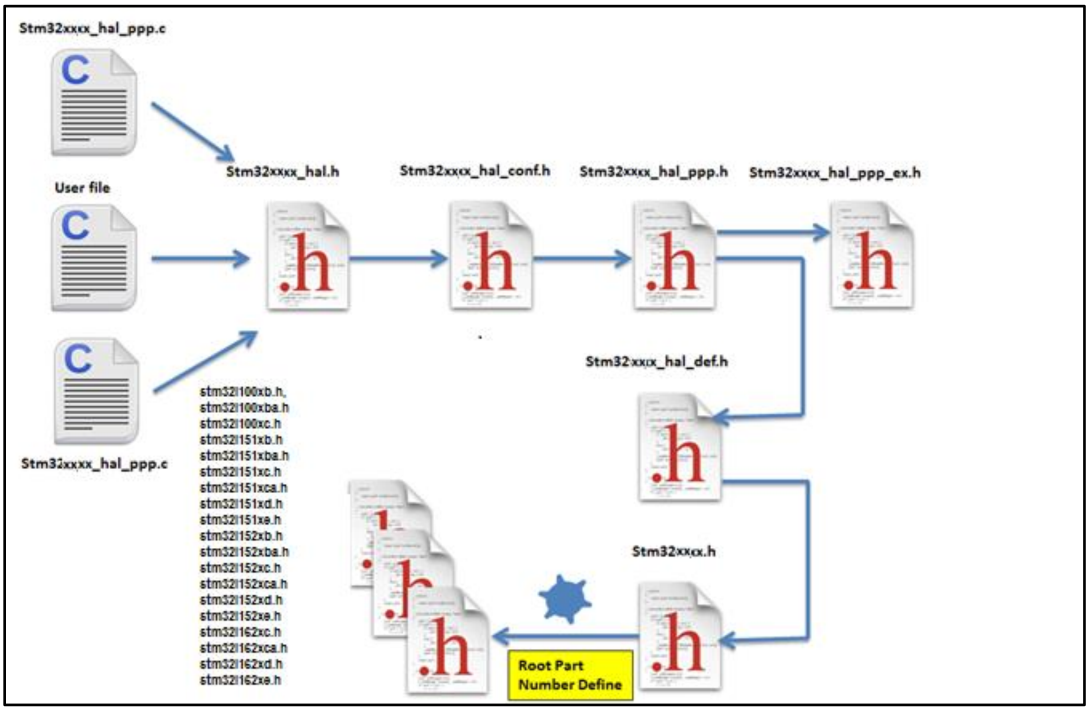
Example:

```
#if defined(STM32F373xC) || defined(STM32F378xx)
typedef struct
{
(...)
}PPP_InitTypeDef;
#endif /* STM32F373xC || STM32F378xx */
```

2.8 File inclusion model

The header of the common HAL driver file (stm32f3xx_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 stm32f3xx_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 *stm32f3xx_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 stm32f3xx_hal_def.h file
calls the stm32f3xx.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 (void *) 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*, stm32f3xx_hal_conf.h,* allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

Table 11: Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 (Hz)
HSE_STARTUP_TIMEOUT	Timeout for HSE start up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
HSI_STARTUP_TIMEOUT	Timeout for HSI start up, expressed in ms	5000
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)
LSI_VALUE	Defines the value of the Internal Low Speed oscillator expressed in Hz. The real value may vary depending on the variations in voltage and temperature.	40 000 (Hz)
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE

The stm32f3xx_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 stm32f3xx_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 and APB 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 stm32f3xx_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, …)
- MCO and CSS configuration functions

A set of macros are defined in stm32f3xx_hal_rcc.h and stm32f3xx_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 stm32f3xx_it.c and implement HAL_GPIO_EXTI_Callback()

The table below describes the GPIO_InitTypeDef structure field.

Table 12: Description of GPIO_InitTypeDef structure

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

Structure field	Description
Alternate	Peripheral to be connected to the selected pins. Possible values: GPIO_AFx_PPP, where AFx: is the alternate function index PPP: is the peripheral instance Example: use GPIO_AF1_TIM2 to connect TIM2 IOs on AF1. These values are defined in the GPIO extended driver, since the AF mapping may change between product lines. Refer to the "Alternate function mapping" table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.

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);
```

- Configuring USART1 Tx (PA9, mapped on AF4) as alternate function:

```
GPIO_InitStruct.Pin = GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
GPIO_InitStruct.Alternate = GPIO_AF4_USART1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, stm32f3xx_hal_cortex.c, provides APIs to handle NVIC and Systick. The supported APIs include:

- HAL_NVIC_SetPriority()
- HAL_NVIC_EnableIRQ()/HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ () / HAL_NVIC_ClearPendingIRQ()
- 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_PVDConfig()
  - 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()
- Backup domain configuration
  - HAL_PWR_EnableBkUpAccess()/ HAL_PWR_DisableBkUpAccess()

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 COMP are configured within the HAL drivers of these peripheral through the macros given in the table below. The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 13: Description of EXTI configuration macros

Macros	Description
PPP_EXTI_LINE_FUNCTION	Defines the EXTI line connected to the internal peripheral. Example: #define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!<External interrupt line 16 Connected to the PVD EXTI Line */
__HAL_PPP_EXTI_ENABLE_IT	Enables a given EXTI line Example: __HAL_PWR_PVD_EXTI_ENABLE_IT()
__HAL_PPP_EXTI_DISABLE_IT	Disables a given EXTI line. Example: __HAL_PWR_PVD_EXTI_DISABLE_IT()
__HAL_PPP_EXTI_GET_FLAG	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PWR_PVD_EXTI_GET_FLAG()

Macros	Description
__HAL_PPP_EXTI_CLEAR_FLAG	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PWR_PVD_EXTI_CLEAR_FLAG()
__HAL_PPP_EXTI_GENERATE_SWIT	Generates a software interrupt for a given EXTI line. Example: __HAL_PWR_PVD_EXTI_GENERATE_SWIT()
__HAL_PPP_EXTI_ENABLE_EVENT	Enables event on a given EXTI Line Example: _HAL_PWR_PVD_EXTI_ENABLE_EVENT()
__HAL_PPP_EXTI_DISABLE_EVENT	Disables event on a given EXTI line Example: __HAL_PWR_PVD_EXTI_DISABLE_EVENT()

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32f3xx_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
- FIFO mode and its Threshold (if needed)
- Burst mode for Source and/or Destination (if needed).

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 channel 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”).

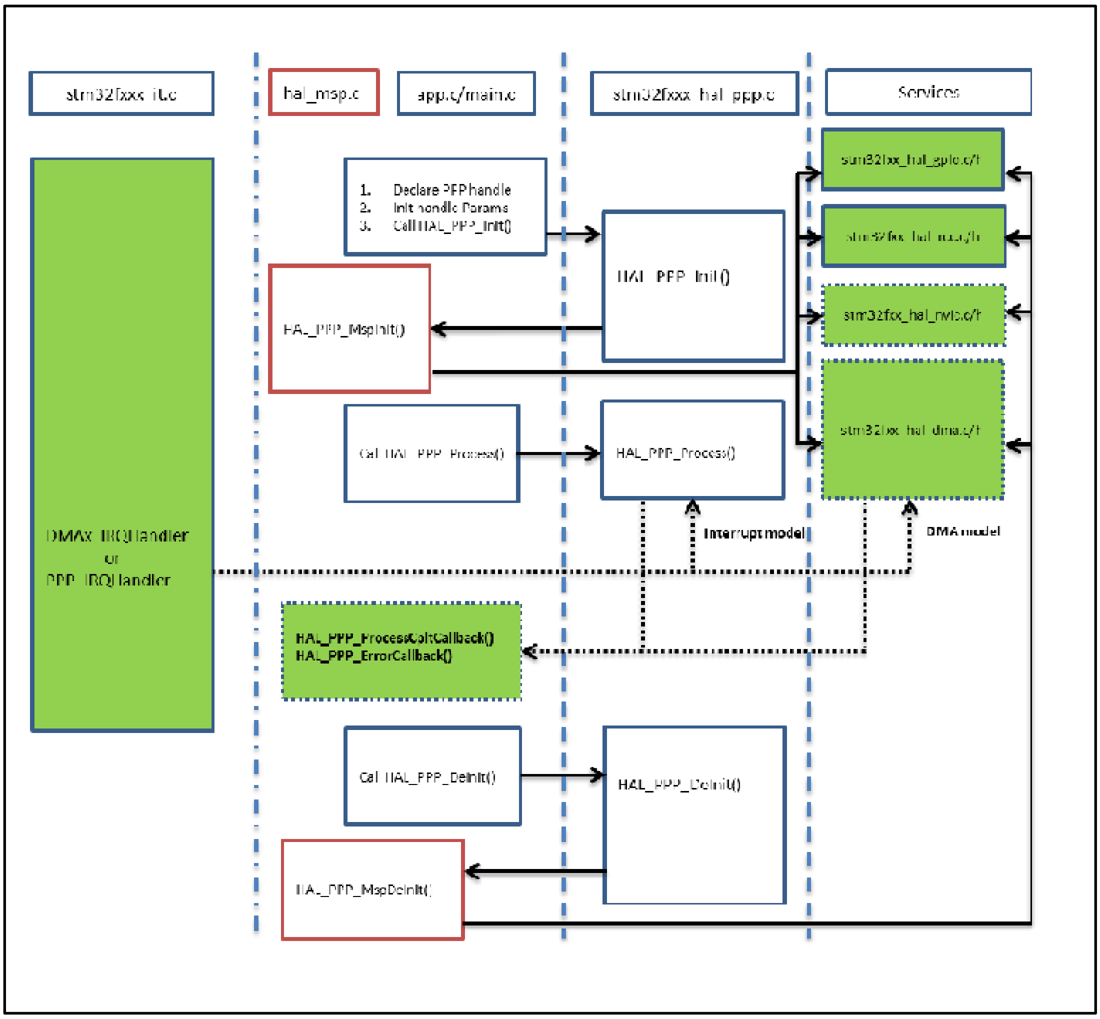
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

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 stm32f3xx_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 RCC_ClkInitStruct;
RCC_OscInitTypeDef RCC_OscInitStruct;
/* Enable HSE Oscillator and activate PLL with HSE as source */
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct)!= HAL_OK)
{
Error_Handler();
}
/* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
clocks dividers */
RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2)!= HAL_OK)
{
Error_Handler();
}
}
```

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 *stm32f3xx_hal_msp.c* file in the user folders. An *stm32f3xx_hal_msp_template.c* file 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.

stm32f3xx_hal_msp.c file contains the following functions:

Table 14: 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 HAL_OK; }
```

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 *stm32f3xx_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)
{
}
```

*stm32f3xx_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 *stm32f3xx_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)
{
}
```

stm32f3xx_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 15: Timeout values

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

Notes:

[(1)]HAL_MAX_DELAY is defined in the stm32fxxx_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 *stm32f3xx_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 stm32f3xx_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:

- HAL Initialization and de-initialization functions
- HAL Control functions

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_EnableDBGSleepMode()***
- ***HAL_DisableDBGSleepMode()***
- ***HAL_EnableDBGStopMode()***
- ***HAL_DisableDBGStopMode()***
- ***HAL_EnableDBGStandbyMode()***
- ***HAL_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.
Return values	• HAL status
Notes	• This function is called at the beginning of program after reset and before the clock configuration • The Systick configuration is based on HSI clock, as HSI is the clock used after a system Reset and the NVIC configuration is set to Priority group 4 • The time base configuration is based on MSI clock when exting from Reset. Once done, time base tick start incrementing. In the default implementation,Systick is used as source of time base. The tick variable is incremented each 1ms in its ISR.

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
Notes	• This function is optional.

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

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

3.1.10 HAL_GetTick

Function Name	**uint32_t HAL_GetTick (void)**
Function Description	Povides a tick value in millisecond.
Return values	• tick value
Notes	• The function is declared as __Weak to be overwritten in case of other implementations in user file.

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
Notes	• In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented. The function is declared as __Weak to be overwritten in case of other implementations in user file.

3.1.12 HAL_SuspendTick

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

3.1.13 HAL_ResumeTick

Function Name	**void HAL_ResumeTick (void)**
Function Description	Resume Tick increment.
Return values	• None
Notes	• In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the the SysTick interrupt will be enabled and so Tick increment is resumed.

The function is declared as __Weak to be overwritten in case of other implementations in user file.

3.1.14 HAL_GetHalVersion

Function Name	**uint32_t HAL_GetHalVersion (void)**
Function Description	This function 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_EnableDBGSleepMode

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

3.1.18 HAL_DisableDBGSleepMode

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

3.1.19 HAL_EnableDBGStopMode

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

3.1.20 HAL_DisableDBGStopMode

Function Name	**void HAL_DisableDBGStopMode (void)**
Function Description	Disable the Debug Module during STOP mode.
Return values	• None

3.1.21 HAL_EnableDBGStandbyMode

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

3.1.22 HAL_DisableDBGStandbyMode

Function Name	**void HAL_DisableDBGStandbyMode (void)**
Function Description	Disable the Debug Module during STANDBY 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 ADC Trigger Remapping

HAL_REMAPADCTRIGGER_ADC12_EXT2	Input trigger of ADC12 regular channel EXT2 0: No remap (TIM1_CC3) 1: Remap (TIM20_TRGO)
HAL_REMAPADCTRIGGER_ADC12_EXT3	Input trigger of ADC12 regular channel EXT3 0: No remap (TIM2_CC2) 1: Remap (TIM20_TRGO2)
HAL_REMAPADCTRIGGER_ADC12_EXT5	Input trigger of ADC12 regular channel EXT5 0: No remap (TIM4_CC4) 1: Remap (TIM20_CC1)
HAL_REMAPADCTRIGGER_ADC12_EXT13	Input trigger of ADC12 regular channel EXT13 0: No remap (TIM6_TRGO) 1: Remap (TIM20_CC2)
HAL_REMAPADCTRIGGER_ADC12_EXT15	Input trigger of ADC12 regular channel EXT15 0: No remap (TIM3_CC4) 1: Remap (TIM20_CC3)
HAL_REMAPADCTRIGGER_ADC12_JEXT3	Input trigger of ADC12 injected channel JEXT3 0: No remap (TIM2_CC1) 1:

	Remap (TIM20_TRGO)
HAL_REMAPADCTRIGGER_ADC12_JEXT6	Input trigger of ADC12 injected channel JEXT6 0: No remap (EXTI line 15) 1: Remap (TIM20_TRGO2)
HAL_REMAPADCTRIGGER_ADC12_JEXT13	Input trigger of ADC12 injected channel JEXT13 0: No remap (TIM3_CC1) 1: Remap (TIM20_CC4)
HAL_REMAPADCTRIGGER_ADC34_EXT5	Input trigger of ADC34 regular channel EXT5 0: No remap (EXTI line 2) 1: Remap (TIM20_TRGO)
HAL_REMAPADCTRIGGER_ADC34_EXT6	Input trigger of ADC34 regular channel EXT6 0: No remap (TIM4_CC1) 1: Remap (TIM20_TRGO2)
HAL_REMAPADCTRIGGER_ADC34_EXT15	Input trigger of ADC34 regular channel EXT15 0: No remap (TIM2_CC1) 1: Remap (TIM20_CC1)
HAL_REMAPADCTRIGGER_ADC34_JEXT5	Input trigger of ADC34 injected channel JEXT5 0: No remap (TIM4_CC3) 1: Remap (TIM20_TRGO)
HAL_REMAPADCTRIGGER_ADC34_JEXT11	Input trigger of ADC34 injected channel JEXT11 0: No remap (TIM1_CC3) 1: Remap (TIM20_TRGO2)
HAL_REMAPADCTRIGGER_ADC34_JEXT14	Input trigger of ADC34 injected channel JEXT14 0: No remap (TIM7_TRGO) 1: Remap (TIM20_CC2)
IS_HAL_REMAPADCTRIGGER	

CRC aliases for Exported Functions

HAL_CRC_Input_Data_Reverse

HAL_CRC_Output_Data_Reverse

HAL DMA Remapping

HAL_REMAPDMA_ADC24_DMA2_CH34	ADC24 DMA remap (STM32F303xB/C/E, STM32F358xx and STM32F398xx devices) 1: Remap (ADC24 DMA requests mapped on DMA2 channels 3 and 4)
HAL_REMAPDMA_TIM16_DMA1_CH6	TIM16 DMA request remap 1: Remap (TIM16_CH1 and TIM16_UP DMA requests mapped on DMA1 channel 6)
HAL_REMAPDMA_TIM17_DMA1_CH7	TIM17 DMA request remap 1: Remap (TIM17_CH1 and TIM17_UP DMA requests mapped on DMA1 channel 7)
HAL_REMAPDMA_TIM6_DAC1_CH1_DMA1_CH3	TIM6 and DAC channel1 DMA remap (STM32F303xB/C/E, STM32F358xx and STM32F398xx

	devices) 1: Remap (TIM6_UP and DAC_CH1 DMA requests mapped on DMA1 channel 3)
HAL_REMAPDMA_TIM7_DAC1_CH2_DMA1_CH4	TIM7 and DAC channel2 DMA remap (STM32F303xB/C/E, STM32F358xx and STM32F398xx devices) 1: Remap (TIM7_UP and DAC_CH2 DMA requests mapped on DMA1 channel 4)
HAL_REMAPDMA_DAC2_CH1_DMA1_CH5	DAC2 channel1 DMA remap (STM32F303x4/6/8 devices only) 1: Remap (DAC2_CH1 DMA requests mapped on DMA1 channel 5)
HAL_REMAPDMA_TIM18_DAC2_CH1_DMA1_CH5	DAC2 channel1 DMA remap (STM32F303x4/6/8 devices only) 1: Remap (DAC2_CH1 DMA requests mapped on DMA1 channel 5)
IS_HAL_REMAPDMA	

HAL I2C Fast Mode Plus

HAL_SYSCFG_FASTMODEPLUS_I2C1	I2C1 fast mode Plus driving capability activation 0: FM+ mode is not enabled on I2C1 pins selected through AF selection bits 1: FM+ mode is enabled on I2C1 pins selected through AF selection bits
HAL_SYSCFG_FASTMODEPLUS_I2C2	I2C2 fast mode Plus driving capability activation 0: FM+ mode is not enabled on I2C2 pins selected through AF selection bits 1: FM+ mode is enabled on I2C2 pins selected through AF selection bits
HAL_SYSCFG_FASTMODEPLUS_I2C3	I2C3 fast mode Plus driving capability activation 0: FM+ mode is not enabled on I2C3 pins selected through AF selection bits 1: FM+ mode is enabled on I2C3 pins selected through AF selection bits
HAL_SYSCFG_FASTMODEPLUS_I2C_PB6	Fast Mode Plus (FM+) driving capability activation on the pad 0: PB6 pin operates in standard mode 1: I2C FM+ mode enabled on PB6 pin, and the Speed control is bypassed
HAL_SYSCFG_FASTMODEPLUS_I2C_PB7	Fast Mode Plus (FM+) driving capability activation on the pad 0: PB7 pin operates in standard mode 1: I2C FM+ mode enabled on PB7 pin, and the Speed control is bypassed
HAL_SYSCFG_FASTMODEPLUS_I2C_PB8	Fast Mode Plus (FM+) driving capability activation on the pad 0: PB8 pin operates in standard mode 1: I2C FM+

	mode enabled on PB8 pin, and the Speed control is bypassed
HAL_SYSCFG_FASTMODEPLUS_I2C_PB9	Fast Mode Plus (FM+) driving capability activation on the pad 0: PB9 pin operates in standard mode 1: I2C FM+ mode enabled on PB9 pin, and the Speed control is bypassed
IS_HAL_SYSCFG_FASTMODEPLUS_CONFIG	

HAL CCM RAM page write protection

HAL_SYSCFG_WP_PAGE0	ICODE SRAM Write protection page 0
HAL_SYSCFG_WP_PAGE1	ICODE SRAM Write protection page 1
HAL_SYSCFG_WP_PAGE2	ICODE SRAM Write protection page 2
HAL_SYSCFG_WP_PAGE3	ICODE SRAM Write protection page 3
HAL_SYSCFG_WP_PAGE4	ICODE SRAM Write protection page 4
HAL_SYSCFG_WP_PAGE5	ICODE SRAM Write protection page 5
HAL_SYSCFG_WP_PAGE6	ICODE SRAM Write protection page 6
HAL_SYSCFG_WP_PAGE7	ICODE SRAM Write protection page 7
HAL_SYSCFG_WP_PAGE8	ICODE SRAM Write protection page 8
HAL_SYSCFG_WP_PAGE9	ICODE SRAM Write protection page 9
HAL_SYSCFG_WP_PAGE10	ICODE SRAM Write protection page 10
HAL_SYSCFG_WP_PAGE11	ICODE SRAM Write protection page 11
HAL_SYSCFG_WP_PAGE12	ICODE SRAM Write protection page 12
HAL_SYSCFG_WP_PAGE13	ICODE SRAM Write protection page 13
HAL_SYSCFG_WP_PAGE14	ICODE SRAM Write protection page 14
HAL_SYSCFG_WP_PAGE15	ICODE SRAM Write protection page 15
IS_HAL_SYSCFG_WP_PAGE	

Constants

__STM32F3xx_HAL_VERSION_MAIN	[31:24] main version
__STM32F3xx_HAL_VERSION_SUB1	[23:16] sub1 version
__STM32F3xx_HAL_VERSION_SUB2	[15:8] sub2 version
__STM32F3xx_HAL_VERSION_RC	[7:0] release candidate
__STM32F3xx_HAL_VERSION	
IDCODE_DEVID_MASK	

HAL SYSCFG Interrupts

HAL_SYSCFG_IT_FPU_IOC	Floating Point Unit Invalid operation Interrupt
HAL_SYSCFG_IT_FPU_DZC	Floating Point Unit Divide-by-zero Interrupt
HAL_SYSCFG_IT_FPU_UFC	Floating Point Unit Underflow Interrupt
HAL_SYSCFG_IT_FPU_OFC	Floating Point Unit Overflow Interrupt

HAL_SYSCFG_IT_FPU_IDC	Floating Point Unit Input denormal Interrupt
HAL_SYSCFG_IT_FPU_IXC	Floating Point Unit Inexact Interrupt
IS_HAL_SYSCFG_INTERRUPT	

HAL Trigger Remapping

HAL_REMAPTRIGGER_DAC1_TRIG	DAC trigger remap (when TSEL = 001 on STM32F303xB/C and STM32F358xx devices) 0: No remap (DAC trigger is TIM8_TRGO) 1: Remap (DAC trigger is TIM3_TRGO)
HAL_REMAPTRIGGER_TIM1_ITR3	TIM1 ITR3 trigger remap 0: No remap 1: Remap (TIM1_TRG3 = TIM17_OC)
IS_HAL_REMAPTRIGGER	

4 HAL ADC Generic Driver

4.1 ADC Firmware driver registers structures

4.1.1 __ADC_HandleTypeDef

__ADC_HandleTypeDef is defined in the stm32f3xx_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

1. 12-bit, 10-bit, 8-bit or 6-bit configurable resolution (available only on STM32F30xxC devices).
2. Interrupt generation at the end of regular conversion, end of injected conversion, and in case of analog watchdog or overrun events.
3. Single and continuous conversion modes.
4. Scan mode for automatic conversion of channel 0 to channel 'n'.

5. Data alignment with in-built data coherency.
6. Channel-wise programmable sampling time.
7. ADC conversion of regular or injected groups.
8. External trigger (timer or EXTI) with configurable polarity for both regular and injected groups.
9. DMA request generation for transfer of conversions data of regular group.
10. Multimode dual mode (available on devices with 2 ADCs or more).
11. Configurable DMA data storage in Multimode Dual mode (available on devices with 2 DCs or more).
12. Configurable delay between conversions in Dual interleaved mode (available on devices with 2 DCs or more).
13. ADC calibration
14. ADC channels selectable single/differential input (available only on STM32F30xxC devices)
15. ADC Injected sequencer&channels configuration context queue (available only on STM32F30xxC devices)
16. ADC offset on injected and regular groups (offset on regular group available only on STM32F30xxC devices)
17. ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
18. 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.
   - For STM32F30x/STM32F33x devices: Two possible clock sources: synchronous clock derived from AHB clock or asynchronous clock derived from ADC dedicated PLL 72MHz. - Synchronous clock is mandatory since used as ADC core clock. Synchronous clock can be used optionally as ADC conversion clock, depending on ADC init structure clock setting. Synchronous clock is configured using macro __ADCx_CLK_ENABLE(). - Asynchronous can be used optionally as ADC conversion clock, depending on ADC init structure clock setting. Asynchronous clock is configured using function HAL_RCCEx_PeriphCLKConfig().
     - For example, in case of device with a single ADC: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC1_CLK_ENABLE() (mandatory)
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC (optional, if ADC conversion from asynchronous clock)
     - PeriphClkInit.Adc1ClockSelection = RCC_ADC1PLLCLK_DIV1 (optional, if ADC conversion from asynchronous clock)
     - HAL_RCCEx_PeriphCLKConfig(&RCC_PeriphClkInitStructure) (optional, if ADC conversion from asynchronous clock)
     - For example, in case of device with 4 ADCs:
     - if((hadc->Instance == ADC1) || (hadc->Instance == ADC2))
     - {
     - __HAL_RCC_ADC12_CLK_ENABLE() (mandatory)

- PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC (optional, if ADC conversion from asynchronous clock)
- PeriphClkInit.Adc12ClockSelection = RCC_ADC12PLLCLK_DIV1 (optional, if ADC conversion from asynchronous clock)
- HAL_RCCEx_PeriphCLKConfig(&RCC_PeriphClkInitStructure) (optional, if ADC conversion from asynchronous clock)
- }
- else
- {
- __HAL_RCC_ADC34_CLK_ENABLE() (mandatory)
- PeriphClkInit.Adc34ClockSelection = RCC_ADC34PLLCLK_DIV1; (optional, if ADC conversion from asynchronous clock)
- HAL_RCCEx_PeriphCLKConfig(&RCC_PeriphClkInitStructure); (optional, if ADC conversion from asynchronous clock)
- }

- For STM32F37x devices: One clock setting is mandatory: ADC clock (core and conversion clock) from APB2 clock.
  - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
  - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC
  - PeriphClkInit.AdcClockSelection = RCC_ADCPLLCLK_DIV2
  - 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()
- HAL_ADCEx_InjectedQueueOverflowCallback() (for STM32F30x/STM32F33x devices)

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.
   - For STM32F30x/STM32F33x devices: Caution: For devices with several ADCs: These settings impact both ADC of common group: ADC1&ADC2, ADC3&ADC4 if available (ADC2, ADC3, ADC4 availability depends on STM32 product)
     - For example, in case of device with a single ADC: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC1_FORCE_RESET() (optional)
     - __HAL_RCC_ADC1_RELEASE_RESET() (optional)
     - __HAL_RCC_ADC1_CLK_DISABLE() (mandatory)
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC (optional, if configured before)
     - PeriphClkInit.Adc1ClockSelection = RCC_ADC1PLLCLK_OFF (optional, if configured before)
     - HAL_RCCEx_PeriphCLKConfig(&RCC_PeriphClkInitStructure) (optional, if configured before)
     - For example, in case of device with 4 ADCs:
     - if((hadc->Instance == ADC1) || (hadc->Instance == ADC2))
     - {
     - __HAL_RCC_ADC12_FORCE_RESET() (optional)
     - __HAL_RCC_ADC12_RELEASE_RESET() (optional)
     - __HAL_RCC_ADC12_CLK_DISABLE() (mandatory)
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC (optional, if configured before)
     - PeriphClkInit.Adc12ClockSelection = RCC_ADC12PLLCLK_OFF (optional, if configured before)
     - HAL_RCCEx_PeriphCLKConfig(&RCC_PeriphClkInitStructure) (optional, if configured before)
     - }
     - else
     - {
     - __HAL_RCC_ADC32_FORCE_RESET() (optional)
     - __HAL_RCC_ADC32_RELEASE_RESET() (optional)
     - __HAL_RCC_ADC34_CLK_DISABLE() (mandatory)

  - PeriphClkInit.Adc34ClockSelection = RCC_ADC34PLLCLK_OFF (optional, if configured before)
  - HAL_RCCEx_PeriphCLKConfig(&RCC_PeriphClkInitStructure) (optional, if configured before)
  - }
- For STM32F37x devices:
  - Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
  - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC
  - PeriphClkInit.AdcClockSelection = RCC_ADCPLLCLK_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 depending on both possible clock sources: PLL clock or AHB clock. 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". • For devices with several ADCs: parameters related to common ADC registers (ADC clock mode) are set only if all ADCs sharing the same common group are disabled. If this is not the case, these common parameters setting are bypassed without error reporting: it can be the intended behaviour in case of update of a parameter of ADC_InitTypeDef on the fly, without disabling the other ADCs sharing the same common group.

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
Notes	• For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behaviour in case of reset of a single ADC while the other ADCs sharing the same common group is still running. • For devices with several ADCs: Global reset of all ADCs sharing a common group is possible. As this function is intended to reset a single ADC, to not impact other ADCs, instructions for global reset of multiple ADCs have been let commented below. If needed, the example code can be copied and uncommented into function HAL_ADC_MspDeInit().

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
Notes	• Case of multimode enabled (for devices with several ADCs): This function must be called for ADC slave first, then ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

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 group 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. • Case of multimode enabled (for devices with several ADCs): This function must be called for ADC master first, then ADC slave. For ADC master, converson is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

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 1 event (main analog watchdog, present on all STM32 devices) ADC_AWD2_EVENT: ADC Analog watchdog 2 event (additional analog watchdog, present only on STM32F3 devices) ADC_AWD3_EVENT: ADC Analog watchdog 3 event (additional analog watchdog, present only on STM32F3 devices) ADC_OVR_EVENT: ADC Overrun event ADC_JQOVF_EVENT: ADC Injected context queue overflow 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 interruption of end-of-conversion, disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• HAL status.
Notes	• ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function. • Case of multimode enabled (for devices with several ADCs): This function must be called for ADC master first, then ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC

slave).

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. • Case of multimode enabled (for devices with several ADCs): This function is for single-ADC mode only. For multimode, use the dedicated MultimodeStop function.

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. Additionally, this functions clears EOS (end of sequence of regular group) flag, in case of the end of the sequence is reached.

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
Notes	• In case of several analog watchdog enabled, if needed to know which one triggered and on which ADCx, Test Analog Watchdog flags ADC_FLAG_AWD1/2/3 into function HAL_ADC_LevelOutOfWindowCallback(). For example: "if (__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_AWD1) != RESET)" "if (__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_AWD2) != RESET)" "if (__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_AWD3) != RESET)"

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. The recommended sampling time is at least: For devices STM32F37x: 17.1us for temperature sensorFor the other STM32F3 devices: 2.2us for each of 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
Notes	• Possibility to update parameters on the fly: This function initializes the selected analog watchdog, following calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" 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_AnalogWDGConfTypeDef".

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 Calibration Factor Length Verification

IS_ADC_CALFACT

Description:

- Calibration factor length verification (7 bits maximum)

Parameters:

- _Calibration_Factor_: Calibration factor value

Return value:

- None:

ADC Conversion Group

ADC_REGULAR_GROUP

ADC_INJECTED_GROUP

ADC_REGULAR_INJECTED_GROUP

ADC Exported Macros

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None:

ADC Injected Conversion Number Verification

IS_ADC_INJECTED_NB_CONV

ADC Multimode Bits

ADC_CCR_MULTI	Multi ADC mode selection
ADC_CCR_MULTI_0	MULTI bit 0
ADC_CCR_MULTI_1	MULTI bit 1
ADC_CCR_MULTI_2	MULTI bit 2
ADC_CCR_MULTI_3	MULTI bit 3
ADC_CCR_MULTI_4	MULTI bit 4
ADC_CCR_DELAY	Delay between 2 sampling phases
ADC_CCR_DELAY_0	DELAY bit 0
ADC_CCR_DELAY_1	DELAY bit 1
ADC_CCR_DELAY_2	DELAY bit 2
ADC_CCR_DELAY_3	DELAY bit 3
ADC_CCR_DMACFG	DMA configuration for multi-ADC mode
ADC_CCR_MDMA	DMA mode for multi-ADC mode
ADC_CCR_MDMA_0	MDMA bit 0
ADC_CCR_MDMA_1	MDMA bit 1
ADC_CCR_CKMODE	ADC clock mode
ADC_CCR_CKMODE_0	CKMODE bit 0
ADC_CCR_CKMODE_1	CKMODE bit 1
ADC_CCR_VREFEN	VREFINT enable
ADC_CCR_TSEN	Temperature sensor enable
ADC_CCR_VBATEN	VBAT enable

ADC Private Constants

ADC_FLAG_ALL

ADC_FLAG_POSTCONV_ALL

ADC Regular Discontinuous Mode NumberVerification

IS_ADC_REGULAR_DISCONT_NUMBER

ADC Regular Conversion Number Verification

IS_ADC_REGULAR_NB_CONV

ADC Trigger Remapping Enable

__HAL_REMAPADCTRIGGER_ENABLE

Description:

- ADC trigger remapping enable/disable macros.

Parameters:

- __ADCTRIGGER_REMAP__: This parameter can be a value of

__HAL_REMAPADCTRIGGER_DISABLE

5 HAL ADC Extension Driver

5.1 ADCEx Firmware driver registers structures

5.1.1 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32f3xx_hal_adc_ex.h

Data Fields

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

Field Documentation

- ***uint32_t ADC_InitTypeDef::ClockPrescaler***
  Select ADC clock source (synchronous clock derived from AHB clock or asynchronous clock derived from ADC dedicated PLL 72MHz) and clock prescaler. The clock is common for all the ADCs. This parameter can be a value of ***ADCEx_ClockPrescaler*** Note: In case of usage of channels on injected group, ADC frequency should be lower than AHB clock frequency /4 for resolution 12 or 10 bits, AHB clock frequency /3 for resolution 8 bits, AHB clock frequency /2 for resolution 6 bits. Note: In case of usage of the ADC dedicated PLL clock, this clock must be preliminarily enabled and prescaler set at RCC top level. Note: This parameter can be modified only if all ADCs of the common ADC group are disabled (for products with several ADCs)
- ***uint32_t ADC_InitTypeDef::Resolution***
  Configures the ADC resolution. This parameter can be a value of ***ADCEx_Resolution***
- ***uint32_t ADC_InitTypeDef::DataAlign***
  Specifies ADC data alignment to right (for resolution 12 bits: MSB on register bit 11 and LSB on register bit 0) (default setting) or to left (for resolution 12 bits, if offset disabled: MSB on register bit 15 and LSB on register bit 4, if offset enabled: MSB on register bit 14 and LSB on register bit 3). See reference manual for alignments with other resolutions. This parameter can be a value of ***ADCEx_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 ***ADCEx_Scan_mode***

- ***uint32_t ADC_InitTypeDef::EOCSelection***
 Specifies what EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of conversion of each rank or complete sequence. This parameter can be a value of ***ADCEx_EOCSelection***.
- ***uint32_t ADC_InitTypeDef::LowPowerAutoWait***
 Selects the dynamic low power Auto Delay: new conversion start only when the previous conversion (for regular group) or previous sequence (for injected group) has been treated by user software. This feature automatically adapts the speed of ADC to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: Do not use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since they have to clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion with **HAL_ADC_Start()**, 2. Later on, when conversion data is needed: use **HAL_ADC_PollForConversion()** to ensure that conversion is completed and use **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion (in case of usage of injected group, use the equivalent functions HAL_ADCExInjected_Start(), **HAL_ADCEx_InjectedGetValue()**, ...).
- ***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 the regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 16. Note: This parameter must be modified when no conversion is on going on regular group (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InitTypeDef::DiscontinuousConvMode***
 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. This parameter can be a value of ***ADCEx_External_trigger_source_Regular*** Caution: For devices with several ADCs, external trigger source is common to ADC common group (for example: ADC1&ADC2, ADC3&ADC4, if available)
- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***
 Selects the external trigger edge of regular group. If trigger is set to

ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ***ADCEx_External_trigger_edge_Regular***

- ***uint32_t ADC_InitTypeDef::DMAContinuousRequests***
  Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***uint32_t ADC_InitTypeDef::Overrun***
  Select the behaviour in case of overrun: data overwritten (default) or preserved. This parameter is for regular group only. This parameter can be a value of ***ADCEx_Overrun*** Note: Case of overrun set to data preserved and usage with end on conversion interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved into function **HAL_ADC_ConvCpltCallback()** (called before end of conversion flags clear). Note: Error reporting in function of conversion mode: Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read the conversion data each time, this is not considered as an erroneous case.Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register, any data missed would be abnormal).

5.1.2 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32f3xx_hal_adc_ex.h

Data Fields

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

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
  Specifies the channel to configure into ADC regular group. This parameter can be a value of ***ADCEx_channels*** Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  Specifies the rank in the regular group sequencer. This parameter can be a value of ***ADCEx_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, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADCEx_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/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS_vrefint, TS_vbat, TS_temp (values rough order: 2.2us min).

- ***uint32_t ADC_ChannelConfTypeDef::SingleDiff***
  Selection of single-ended or differential input. In differential mode: Differential measurement is between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADCEx_SingleDifferential*** Caution: This parameter 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: Channels 1 to 14 are available in differential mode. Channels 15, 16, 17, 18 can be used only in single-ended mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behaviour in case of another parameter update on the fly)
- ***uint32_t ADC_ChannelConfTypeDef::OffsetNumber***
  Selects the offset number This parameter can be a value of ***ADCEx_OffsetNumber*** Caution: Only one channel is allowed per channel. If another channel was on this offset number, the offset will be changed to the new channel
- ***uint32_t ADC_ChannelConfTypeDef::Offset***
  Defines the offset to be subtracted from the raw converted data when convert channels. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

5.1.3 ADC_InjectionConfTypeDef

ADC_InjectionConfTypeDef is defined in the stm32f3xx_hal_adc_ex.h

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedSingleDiff***
- ***uint32_t InjectedOffsetNumber***
- ***uint32_t InjectedOffset***
- ***uint32_t InjectedNbrOfConversion***
- ***uint32_t InjectedDiscontinuousConvMode***
- ***uint32_t AutoInjectedConv***
- ***uint32_t QueueInjectedContext***
- ***uint32_t ExternalTrigInjecConv***

- ***uint32_t ExternalTrigInjecConvEdge***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  Configure the ADC injected channel This parameter can be a value of ***ADCEx_channels*** Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  The rank in the regular 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, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADCEx_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/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS_vrefint, TS_vbat, TS_temp (values rough order: 2.2us min).
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSingleDiff***
  Selection of single-ended or differential input. In differential mode: Differential measurement is between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADCEx_SingleDifferential*** Caution: This parameter 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: Channels 1 to 14 are available in differential mode. Channels 15, 16, 17, 18 can be used only in single-ended mode. Note: When configuring a channel 'i' in differential mode, the channel 'i-1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behaviour in case of another parameter update on the fly)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetNumber***
  Selects the offset number This parameter can be a value of ***ADCEx_OffsetNumber*** Caution: Only one channel is allowed per offset number. If another channel was on this offset number, the offset will be changed to the new channel.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***
  Defines the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.
- ***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: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). 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::QueueInjectedContext***
  Specifies whether the context queue feature is enabled. This parameter can be set to ENABLE or DISABLE. If context queue is enabled, injected sequencer&channels configurations are queued on up to 2 contexts. If a new injected context is set when queue is full, error is triggered by interruption and through function 'HAL_ADCEx_InjectedQueueOverflowCallback'. Caution: This feature request that the sequence is fully configured before injected conversion start. Therefore, configure channels with **HAL_ADCEx_InjectedConfigChannel()** as many times as value of 'InjectedNbrOfConversion' parameter. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion).
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***
  Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled. This parameter can be a value of ***ADCEx_External_trigger_source_Injected*** Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***
  Selects the external trigger edge of injected group. This parameter can be a value of ***ADCEx_External_trigger_edge_Injected***. If trigger is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

5.1.4 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32f3xx_hal_adc_ex.h

Data Fields

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

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***
  Selects which ADC analog watchdog to apply to the selected channel. For Analog Watchdog 1: Only 1 channel can be monitored (or overall group of channels by setting parameter 'WatchdogMode') For Analog Watchdog 2 and 3: Several channels can be monitored (by successive calls of '**HAL_ADC_AnalogWDGConfig()**' for each channel) This parameter can be a value of ***ADCEx_analog_watchdog_number***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***
  For Analog Watchdog 1: Configures the ADC analog watchdog mode: single channel/overall group of channels, regular/injected group. For Analog Watchdog 2 and 3: There is no configuration for overall group of channels as AWD1. Set value 'ADC_ANALOGWATCHDOG_NONE' to reset channels group programmed with parameter 'Channel', set any other value to not use this parameter. This parameter can be a value of ***ADCEx_analog_watchdog_mode***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***
  Selects which ADC channel to monitor by analog watchdog. For Analog Watchdog 1: this parameter has an effect only if parameter 'WatchdogMode' is configured on single channel. Only 1 channel can be monitored. For Analog Watchdog 2 and 3: Several channels can be monitored (successive calls of **HAL_ADC_AnalogWDGConfig()** must be done, one for each channel. Channels group reset can be done by setting WatchdogMode to 'ADC_ANALOGWATCHDOG_NONE'). This parameter can be a value of ***ADCEx_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. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  Configures the ADC analog watchdog High threshold value. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.

5.1.5 ADC_MultiModeTypeDef

ADC_MultiModeTypeDef is defined in the stm32f3xx_hal_adc_ex.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t DMAAccessMode***
- ***uint32_t TwoSamplingDelay***

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***
- ***uint32_t ADC_MultiModeTypeDef::DMAAccessMode***
  Configures the DMA mode for multi ADC mode: selection whether 2 DMA channels (each ADC use its own DMA channel) or 1 DMA channel (one DMA channel for both ADC, DMA of ADC master) This parameter can be a value of ***ADCEx_Direct_memory_access_mode_for_multimode*** Caution: Limitations with multimode DMA access enabled (1 DMA channel used): In case of dual mode in high speed (more than 5Msps) or high activity of DMA by other peripherals, there is a risk of DMA overrun. Therefore, it is recommended to disable multimode DMA access: each ADC uses its own DMA channel. Refer to device errata sheet for more details.
- ***uint32_t ADC_MultiModeTypeDef::TwoSamplingDelay***
  Configures the Delay between 2 sampling phases. This parameter can be a value of ***ADCEx_delay_between_2_sampling_phases*** Delay range depends on selected resolution: from 1 to 12 clock cycles for 12 bits, from 1 to 10 clock cycles for 10 bits from 1 to 8 clock cycles for 8 bits, from 1 to 6 clock cycles for 6 bits

5.2 ADCEx Firmware driver API description

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

5.2.1 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()***

5.2.2 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.
- 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_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_ADCEx_Calibration_Start()*
- *HAL_ADCEx_Calibration_GetValue()*
- *HAL_ADCEx_Calibration_SetValue()*
- *HAL_ADCEx_InjectedStart()*
- *HAL_ADCEx_InjectedStop()*
- *HAL_ADCEx_InjectedPollForConversion()*
- *HAL_ADCEx_InjectedStart_IT()*
- *HAL_ADCEx_InjectedStop_IT()*
- *HAL_ADCEx_MultiModeStart_DMA()*
- *HAL_ADCEx_MultiModeStop_DMA()*
- *HAL_ADCEx_MultiModeGetValue()*
- *HAL_ADCEx_InjectedGetValue()*
- *HAL_ADCEx_InjectedConvCpltCallback()*
- *HAL_ADCEx_InjectedQueueOverflowCallback()*

5.2.3 Peripheral Control functions

This section provides functions allowing to:

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

- *HAL_ADCEx_InjectedConfigChannel()*
- *HAL_ADC_AnalogWDGConfig()*
- *HAL_ADCEx_MultiModeConfigChannel()*

5.2.4 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 depending on both possible clock sources: PLL clock or AHB clock. 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". • For devices with several ADCs: parameters related to common ADC registers (ADC clock mode) are set only if all ADCs sharing the same common group are disabled. If this is not the case, these common parameters setting are bypassed without error reporting: it can be the intended behaviour in case of update of a parameter of ADC_InitTypeDef on the fly, without disabling the other ADCs sharing the same common group.

5.2.5 HAL_ADC_DeInit

Function Name	**HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)**
Function Description	Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.
Parameters	• **hadc:** ADC handle
Return values	• HAL status
Notes	• For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common

group are disabled. If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behaviour in case of reset of a single ADC while the other ADCs sharing the same common group is still running.
- For devices with several ADCs: Global reset of all ADCs sharing a common group is possible. As this function is intended to reset a single ADC, to not impact other ADCs, instructions for global reset of multiple ADCs have been let commented below. If needed, the example code can be copied and uncommented into function HAL_ADC_MspDeInit().

5.2.6 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
Notes	• Case of multimode enabled (for devices with several ADCs): This function must be called for ADC slave first, then ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

5.2.7 HAL_ADC_Stop

Function Name	**HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)**
Function Description	Stop ADC conversion of regular group (and injected group 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. • Case of multimode enabled (for devices with several ADCs): This function must be called for ADC master first, then ADC slave. For ADC master, converson is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

5.2.8 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

5.2.9 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 1 event (main analog watchdog, present on all STM32 devices) ADC_AWD2_EVENT: ADC Analog watchdog 2 event (additional analog watchdog, present only on STM32F3 devices) ADC_AWD3_EVENT: ADC Analog watchdog 3 event (additional analog watchdog, present only on STM32F3 devices) ADC_OVR_EVENT: ADC Overrun event ADC_JQOVF_EVENT: ADC Injected context queue overflow event • **Timeout:** Timeout value in millisecond.
Return values	• HAL status

5.2.10 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.

5.2.11 HAL_ADC_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)**
Function Description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interruption of end-of-conversion, disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• HAL status.
Notes	• ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be

preliminarily stopped using HAL_ADCEx_InjectedStop function.

- Case of multimode enabled (for devices with several ADCs): This function must be called for ADC master first, then ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

5.2.12 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.

5.2.13 HAL_ADC_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)**
Function Description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.
Parameters	• **hadc:** ADC handle
Return values	• HAL status.
Notes	• ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function. • Case of multimode enabled (for devices with several ADCs): This function is for single-ADC mode only. For multimode, use the dedicated MultimodeStop function.

5.2.14 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. Additionally, this functions clears EOS (end of sequence of regular group) flag, in case of the end of the sequence is reached.

5.2.15 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

5.2.16 HAL_ADCEx_Calibration_Start

Function Name	**HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)**
Function Description	Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop()).
Parameters	• **hadc:** ADC handle • **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values: ADC_SINGLE_ENDED: Channel in mode input single ended ADC_DIFFERENTIAL_ENDED: Channel in mode input differential ended
Return values	• HAL status

5.2.17 HAL_ADCEx_Calibration_GetValue

Function Name	**uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)**
Function Description	Get the calibration factor from automatic conversion result.
Parameters	• **hadc:** ADC handle • **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values: ADC_SINGLE_ENDED: Channel in mode input single ended ADC_DIFFERENTIAL_ENDED: Channel in mode input differential ended
Return values	• Converted value

5.2.18 HAL_ADCEx_Calibration_SetValue

Function Name	**HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)**
Function Description	Set the calibration factor to overwrite automatic conversion result.
Parameters	• **hadc:** ADC handle • **SingleDiff:** Selection of single-ended or differential input This

	parameter can be one of the following values: ADC_SINGLE_ENDED: Channel in mode input single ended ADC_DIFFERENTIAL_ENDED: Channel in mode input differential ended • **CalibrationFactor:** Calibration factor (coded on 7 bits maximum)
Return values	• HAL state

5.2.19 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
Notes	• Case of multimode enabled (for devices with several ADCs): This function must be called for ADC slave first, then ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

5.2.20 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. • Case of multimode enabled (for devices with several ADCs): This function must be called for ADC master first, then ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

5.2.21 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

5.2.22 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.

5.2.23 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. • Case of multimode enabled (for devices with several ADCs): This function must be called for ADC master first, then ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave). • In case of auto-injection mode, HAL_ADC_Stop must be used.

5.2.24 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.

5.2.25 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.

5.2.26 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.

5.2.27 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

5.2.28 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

5.2.29 HAL_ADCEx_InjectedQueueOverflowCallback

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

5.2.30 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. The recommended sampling time is at least: For devices STM32F37x: 17.1us for temperature sensorFor the other STM32F3 devices: 2.2us for each of 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".

5.2.31 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. For parameters constraints, see comments of structure "ADC_InjectionConfTypeDef". • In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. The recommended sampling time is at least: For devices STM32F37x: 17.1us for temperature sensorFor the other STM32F3 devices: 2.2us for each of channels Vbat/VrefInt/TempSensor. These internal paths can be be disabled using function HAL_ADC_DeInit(). • To reset injected sequencer, function HAL_ADCEx_InjectedStop() can be used. • Caution: For Injected Context Queue use: a context must be fully defined before start of injected conversion: all channels configured consecutively for the same ADC instance. Therefore, Number of calls of HAL_ADCEx_InjectedConfigChannel() must correspond to value of parameter InjectedNbrOfConversion for each context. Example 1: If 1 context intended to be used (or not use of this feature: QueueInjectedContext=DISABLE) and usage of the 3 first injected ranks (InjectedNbrOfConversion=3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel (3 times) before launching a conversion. This function must not be called to configure the 4th injected channel: it would start a new context into context queue.Example 2: If 2 contexts intended to be used and usage of the 3 first injected ranks (InjectedNbrOfConversion=3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel and for each context (3 channels x 2 contexts = 6 calls). Conversion can start once the 1st context is set. The 2nd context can be set on the fly.

5.2.32 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
Notes	•	Possibility to update parameters on the fly: This function initializes the selected analog watchdog, following calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" 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_AnalogWDGConfTypeDef".

5.2.33 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 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 Extended Analog Watchdog Selection

ADC_ANALOGWATCHDOG_1

ADC_ANALOGWATCHDOG_2

ADC_ANALOGWATCHDOG_3

ADC Extended Channels

ADC_CHANNEL_1

ADC_CHANNEL_2

ADC_CHANNEL_3

ADC_CHANNEL_4

ADC_CHANNEL_5

ADC_CHANNEL_6

ADC_CHANNEL_7

ADC_CHANNEL_8

ADC_CHANNEL_9

ADC_CHANNEL_10

ADC_CHANNEL_11

ADC_CHANNEL_12

ADC_CHANNEL_13

ADC_CHANNEL_14

ADC_CHANNEL_15

ADC_CHANNEL_16

ADC_CHANNEL_17

ADC_CHANNEL_18

ADC_CHANNEL_VOPAMP1

ADC_CHANNEL_TEMPSENSOR

ADC_CHANNEL_VBAT

ADC_CHANNEL_VOPAMP2

ADC_CHANNEL_VOPAMP3

ADC_CHANNEL_VOPAMP4

ADC_CHANNEL_VREFINT

ADC Extended Clock Prescaler

ADC_CLOCK_ASYNC_DIV1	ADC asynchronous clock derived from ADC dedicated PLL
ADC_CLOCK_SYNC_PCLK_DIV1	ADC synchronous clock derived from AHB clock without prescaler
ADC_CLOCK_SYNC_PCLK_DIV2	ADC synchronous clock derived from AHB clock divided by a prescaler of 2
ADC_CLOCK_SYNC_PCLK_DIV4	ADC synchronous clock derived from AHB clock divided by a prescaler of 4

IS_ADC_CLOCKPRESCALER

ADC Extended Dual ADC Mode

ADC_MODE_INDEPENDENT

ADC_DUALMODE_REGSIMULT_INJECSIMULT

ADC_DUALMODE_REGSIMULT_ALTERTRIG

ADC_DUALMODE_REGINTERL_INJECSIMULT

ADC_DUALMODE_INJECSIMULT

ADC_DUALMODE_REGSIMULT

ADC_DUALMODE_INTERL

ADC_DUALMODE_ALTERTRIG

ADC Extended Data Alignment

ADC_DATAALIGN_RIGHT

ADC_DATAALIGN_LEFT

ADC Extended Delay Between 2 Sampling Phases

ADC_TWOSAMPLINGDELAY_1CYCLE

ADC_TWOSAMPLINGDELAY_2CYCLES

ADC_TWOSAMPLINGDELAY_3CYCLES

ADC_TWOSAMPLINGDELAY_4CYCLES

ADC_TWOSAMPLINGDELAY_5CYCLES

ADC_TWOSAMPLINGDELAY_6CYCLES

ADC_TWOSAMPLINGDELAY_7CYCLES

ADC_TWOSAMPLINGDELAY_8CYCLES

ADC_TWOSAMPLINGDELAY_9CYCLES

ADC_TWOSAMPLINGDELAY_10CYCLES

ADC_TWOSAMPLINGDELAY_11CYCLES

ADC_TWOSAMPLINGDELAY_12CYCLES

ADC Extended DMA Mode for Dual ADC Mode

ADC_DMAACCESSMODE_DISABLED	DMA multimode disabled: each ADC will use its own DMA channel
ADC_DMAACCESSMODE_12_10_BITS	DMA multimode enabled (one DMA channel for both ADC, DMA of ADC master) for 12 and 10 bits resolution
ADC_DMAACCESSMODE_8_6_BITS	DMA multimode enabled (one DMA channel for both ADC, DMA of ADC master) for 8 and 6 bits resolution

ADC Extended End of Regular Sequence/Conversion

ADC_EOC_SINGLE_CONV

ADC_EOC_SEQ_CONV

ADC_EOC_SINGLE_SEQ_CONV	reserved for future use

ADC Extended 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
HAL_ADC_ERROR_JQOVF	Injected context queue overflow error

ADC Extended Event Type

ADC_AWD1_EVENT	ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 devices)
ADC_AWD2_EVENT	ADC Analog watchdog 2 event (additional analog watchdog, present only on STM32F3 devices)
ADC_AWD3_EVENT	ADC Analog watchdog 3 event (additional analog watchdog, present only on STM32F3 devices)
ADC_OVR_EVENT	ADC overrun event
ADC_JQOVF_EVENT	ADC Injected Context Queue Overflow event
ADC_AWD_EVENT	

ADCEx 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_RDY: ADC Ready (ADRDY) interrupt source
- ADC_IT_EOSMP: ADC End of Sampling interrupt source
- ADC_IT_EOC: ADC End of Regular Conversion interrupt source
- ADC_IT_EOS: ADC End of Regular sequence of Conversions interrupt source
- ADC_IT_OVR: ADC overrun interrupt source
- ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
- ADC_IT_JEOS: ADC End of Injected sequence of Conversions interrupt source
- ADC_IT_AWD1: ADC Analog watchdog 1 interrupt source (main analog watchdog, present on all STM32 devices)
- ADC_IT_AWD2: ADC Analog watchdog 2 interrupt source (additional analog watchdog, present only on STM32F3 devices)
- ADC_IT_AWD3: ADC Analog watchdog 3 interrupt source (additional analog watchdog, present only on STM32F3 devices)
- ADC_IT_JQOVF: ADC Injected Context Queue Overflow 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_RDY: ADC Ready (ADRDY) interrupt source
  - ADC_IT_EOSMP: ADC End of Sampling interrupt source
  - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS: ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR: ADC overrun interrupt

source
  - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS: ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1: ADC Analog watchdog 1 interrupt source (main analog watchdog, present on all STM32 devices)
  - ADC_IT_AWD2: ADC Analog watchdog 2 interrupt source (additional analog watchdog, present only on STM32F3 devices)
  - ADC_IT_AWD3: ADC Analog watchdog 3 interrupt source (additional analog watchdog, present only on STM32F3 devices)
  - ADC_IT_JQOVF: ADC Injected Context Queue Overflow interrupt source

Return value:

- None:

__HAL_ADC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be any combination of the following values:
  - ADC_IT_RDY: ADC Ready (ADRDY) interrupt source
  - ADC_IT_EOSMP: ADC End of Sampling interrupt source
  - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS: ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR: ADC overrun interrupt source
  - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS: ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1: ADC Analog watchdog 1 interrupt source (main analog watchdog, present on all STM32 devices)

- ADC_IT_AWD2: ADC Analog watchdog 2 interrupt source (additional analog watchdog, present only on STM32F3 devices)
- ADC_IT_AWD3: ADC Analog watchdog 3 interrupt source (additional analog watchdog, present only on STM32F3 devices)
- ADC_IT_JQOVF: ADC Injected Context Queue Overflow interrupt source

Return value:

- State: of interruption (SET or RESET)

__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_RDY: ADC Ready (ADRDY) flag
  - ADC_FLAG_EOSMP: ADC End of Sampling flag
  - ADC_FLAG_EOC: ADC End of Regular Conversion flag
  - ADC_FLAG_EOS: ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR: ADC overrun flag
  - ADC_FLAG_JEOC: ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS: ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1: ADC Analog watchdog 1 flag (main analog watchdog, present on all STM32 devices)
  - ADC_FLAG_AWD2: ADC Analog watchdog 2 flag (additional analog watchdog, present only on STM32F3 devices)
  - ADC_FLAG_AWD3: ADC Analog watchdog 3 flag (additional analog watchdog, present only on STM32F3 devices)
  - ADC_FLAG_JQOVF: ADC Injected Context Queue Overflow 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_RDY: ADC Ready (ADRDY) flag
  - ADC_FLAG_EOSMP: ADC End of Sampling flag
  - ADC_FLAG_EOC: ADC End of Regular Conversion flag
  - ADC_FLAG_EOS: ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR: ADC overrun flag
  - ADC_FLAG_JEOC: ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS: ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1: ADC Analog watchdog 1 flag (main analog watchdog, present on all STM32 devices)
  - ADC_FLAG_AWD2: ADC Analog watchdog 2 flag (additional analog watchdog, present only on STM32F3 devices)
  - ADC_FLAG_AWD3: ADC Analog watchdog 3 flag (additional analog watchdog, present only on STM32F3 devices)
  - ADC_FLAG_JQOVF: ADC Injected Context Queue Overflow flag

Return value:

- None:

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None:

External Trigger Edge of Injected Group

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE

ADC_EXTERNALTRIGINJECCONV_EDGE_RISING

ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING

ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING

ADC Extended External trigger enable and polarity selection for regular group

ADC_EXTERNALTRIGCONVEDGE_NONE

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC_EXTERNALTRIGCONVEDGE_FALLING

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

External Trigger Source of Injected Group

ADC_EXTERNALTRIGINJECCONV_T2_CC1

ADC_EXTERNALTRIGINJECCONV_T3_CC1

ADC_EXTERNALTRIGINJECCONV_T3_CC3

ADC_EXTERNALTRIGINJECCONV_T3_CC4

ADC_EXTERNALTRIGINJECCONV_T6_TRGO

ADC_EXTERNALTRIGINJECCONV_EXT_IT15

ADC_EXTERNALTRIGINJECCONV_T1_CC3

ADC_EXTERNALTRIGINJECCONV_T4_CC3

ADC_EXTERNALTRIGINJECCONV_T4_CC4

ADC_EXTERNALTRIGINJECCONV_T7_TRGO

ADC_EXTERNALTRIGINJECCONV_T8_CC2

ADC_EXTERNALTRIGINJECCONV_T1_CC4

ADC_EXTERNALTRIGINJECCONV_T1_TRGO

ADC_EXTERNALTRIGINJECCONV_T1_TRGO2

ADC_EXTERNALTRIGINJECCONV_T2_TRGO

ADC_EXTERNALTRIGINJECCONV_T3_TRGO

ADC_EXTERNALTRIGINJECCONV_T4_TRGO

ADC_EXTERNALTRIGINJECCONV_T8_CC4

ADC_EXTERNALTRIGINJECCONV_T8_TRGO

ADC_EXTERNALTRIGINJECCONV_T8_TRGO2

ADC_EXTERNALTRIGINJECCONV_T15_TRGO

ADC_INJECTED_SOFTWARE_START

ADC_EXTERNALTRIGINJECCONV_T20_CC4	External trigger of injected group for ADC1&ADC2 only, specific to device STM303xE, using Timer20 with ADC trigger input remap. Remap trigger using macro
ADC_EXTERNALTRIGINJECCONV_T20_CC2	External trigger of injected group for ADC1&ADC2 only, specific to device STM303xE, using Timer20 with ADC trigger input remap. Remap trigger using macro
ADC_EXTERNALTRIGINJECCONV_T20_TRGO	External trigger of regular group for ADC1&ADC2, ADC3&ADC4, specific to device STM303xE, using Timer20

	with ADC trigger input remap. For ADC1&ADC2: Remap trigger using macro
ADC_EXTERNALTRIGINJECCONV_T20_TRGO2	External trigger of regular group for ADC1&ADC2, ADC3&ADC4, specific to device STM303xE, using Timer20 with ADC trigger input remap. For ADC1&ADC2: Remap trigger using macro

ADC Extended External trigger selection for regular group

ADC_EXTERNALTRIGCONV_T1_CC1	
ADC_EXTERNALTRIGCONV_T1_CC2	
ADC_EXTERNALTRIGCONV_T2_CC2	
ADC_EXTERNALTRIGCONV_T3_CC4	
ADC_EXTERNALTRIGCONV_T4_CC4	
ADC_EXTERNALTRIGCONV_T6_TRGO	
ADC_EXTERNALTRIGCONV_EXT_IT11	
ADC_EXTERNALTRIGCONV_T2_CC1	
ADC_EXTERNALTRIGCONV_T2_CC3	
ADC_EXTERNALTRIGCONV_T3_CC1	
ADC_EXTERNALTRIGCONV_T4_CC1	
ADC_EXTERNALTRIGCONV_T7_TRGO	
ADC_EXTERNALTRIGCONV_T8_CC1	
ADC_EXTERNALTRIGCONV_EXT_IT2	
ADC_EXTERNALTRIGCONV_T1_CC3	
ADC_EXTERNALTRIGCONV_T1_TRGO	
ADC_EXTERNALTRIGCONV_T1_TRGO2	
ADC_EXTERNALTRIGCONV_T2_TRGO	
ADC_EXTERNALTRIGCONV_T3_TRGO	
ADC_EXTERNALTRIGCONV_T4_TRGO	
ADC_EXTERNALTRIGCONV_T8_TRGO	
ADC_EXTERNALTRIGCONV_T8_TRGO2	
ADC_EXTERNALTRIGCONV_T15_TRGO	
ADC_SOFTWARE_START	
ADC_EXTERNALTRIGCONV_T20_MASK	
ADC_EXTERNALTRIGCONV_T20_CC2	External trigger of regular group for ADC1&ADC2 only, specific to device STM303xE, using Timer20 with ADC trigger input remap. Remap trigger using macro
ADC_EXTERNALTRIGCONV_T20_CC3	External trigger of regular group for

	ADC1&ADC2 only, specific to device STM303xE, using Timer20 with ADC trigger input remap. Remap trigger using macro
ADC_EXTERNALTRIGCONV_T20_CC1	External trigger of regular group for ADC1&ADC2, ADC3&ADC4, specific to device STM303xE, sing Timer20 with ADC trigger input remap. For ADC1&ADC2: Remap trigger using macro
ADC_EXTERNALTRIGCONV_T20_TRGO	External trigger of regular group for ADC1&ADC2, ADC3&ADC4, specific to device STM303xE, sing Timer20 with ADC trigger input remap. For ADC1&ADC2: Remap trigger using macro
ADC_EXTERNALTRIGCONV_T20_TRGO2	External trigger of regular group for ADC1&ADC2, ADC3&ADC4, specific to device STM303xE, sing Timer20 with ADC trigger input remap. For ADC1&ADC2: Remap trigger using macro

ADC Extended Flags Definition

ADC_FLAG_RDY	ADC Ready (ADRDY) flag
ADC_FLAG_EOSMP	ADC End of Sampling flag
ADC_FLAG_EOC	ADC End of Regular Conversion flag
ADC_FLAG_EOS	ADC End of Regular sequence of Conversions flag
ADC_FLAG_OVR	ADC overrun flag
ADC_FLAG_JEOC	ADC End of Injected Conversion flag
ADC_FLAG_JEOS	ADC End of Injected sequence of Conversions flag
ADC_FLAG_AWD1	ADC Analog watchdog 1 flag (main analog watchdog, present on all STM32 devices)
ADC_FLAG_AWD2	ADC Analog watchdog 2 flag (additional analog watchdog, present only on STM32F3 devices)
ADC_FLAG_AWD3	ADC Analog watchdog 3 flag (additional analog watchdog, present only on STM32F3 devices)
ADC_FLAG_JQOVF	ADC Injected Context Queue Overflow flag
ADC_FLAG_AWD	

ADC Extended Injected Channel Rank

ADC_INJECTED_RANK_1

ADC_INJECTED_RANK_2

ADC_INJECTED_RANK_3

ADC_INJECTED_RANK_4

ADC Extended External Trigger Source of Injected Group (Internal)

ADC1_2_EXTERNALTRIGINJEC_T1_TRGO

ADC1_2_EXTERNALTRIGINJEC_T1_CC4

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

ADC1_2_EXTERNALTRIGINJEC_T1_TRGO2

ADC1_2_EXTERNALTRIGINJEC_T8_TRGO

ADC1_2_EXTERNALTRIGINJEC_T8_TRGO2

ADC1_2_EXTERNALTRIGINJEC_T3_CC3

ADC1_2_EXTERNALTRIGINJEC_T3_TRGO

ADC1_2_EXTERNALTRIGINJEC_T3_CC1

ADC1_2_EXTERNALTRIGINJEC_T6_TRGO

ADC1_2_EXTERNALTRIGINJEC_T15_TRGO

ADC3_4_EXTERNALTRIGINJEC_T1_TRGO

ADC3_4_EXTERNALTRIGINJEC_T1_CC4

ADC3_4_EXTERNALTRIGINJEC_T4_CC3

ADC3_4_EXTERNALTRIGINJEC_T8_CC2

ADC3_4_EXTERNALTRIGINJEC_T8_CC4

ADC3_4_EXTERNALTRIGINJEC_T20_TRGO

ADC3_4_EXTERNALTRIGINJEC_T4_CC4

ADC3_4_EXTERNALTRIGINJEC_T4_TRGO

ADC3_4_EXTERNALTRIGINJEC_T1_TRGO2

ADC3_4_EXTERNALTRIGINJEC_T8_TRGO

ADC3_4_EXTERNALTRIGINJEC_T8_TRGO2

ADC3_4_EXTERNALTRIGINJEC_T1_CC3

ADC3_4_EXTERNALTRIGINJEC_T3_TRGO

ADC3_4_EXTERNALTRIGINJEC_T2_TRGO

ADC3_4_EXTERNALTRIGINJEC_T7_TRGO

ADC3_4_EXTERNALTRIGINJEC_T15_TRGO

ADC Extended External trigger selection for regular group (Used Internally)

ADC1_2_EXTERNALTRIG_T1_CC1

ADC1_2_EXTERNALTRIG_T1_CC2

ADC1_2_EXTERNALTRIG_T1_CC3

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

ADC1_2_EXTERNALTRIG_T8_TRGO2

ADC1_2_EXTERNALTRIG_T1_TRGO

ADC1_2_EXTERNALTRIG_T1_TRGO2

ADC1_2_EXTERNALTRIG_T2_TRGO

ADC1_2_EXTERNALTRIG_T4_TRGO

ADC1_2_EXTERNALTRIG_T6_TRGO

ADC1_2_EXTERNALTRIG_T15_TRGO

ADC1_2_EXTERNALTRIG_T3_CC4

ADC3_4_EXTERNALTRIG_T3_CC1

ADC3_4_EXTERNALTRIG_T2_CC3

ADC3_4_EXTERNALTRIG_T1_CC3

ADC3_4_EXTERNALTRIG_T8_CC1

ADC3_4_EXTERNALTRIG_T8_TRGO

ADC3_4_EXTERNALTRIG_EXT_IT2

ADC3_4_EXTERNALTRIG_T4_CC1

ADC3_4_EXTERNALTRIG_T2_TRGO

ADC3_4_EXTERNALTRIG_T8_TRGO2

ADC3_4_EXTERNALTRIG_T1_TRGO

ADC3_4_EXTERNALTRIG_T1_TRGO2

ADC3_4_EXTERNALTRIG_T3_TRGO

ADC3_4_EXTERNALTRIG_T4_TRGO

ADC3_4_EXTERNALTRIG_T7_TRGO

ADC3_4_EXTERNALTRIG_T15_TRGO

ADC3_4_EXTERNALTRIG_T2_CC1

ADC Extended Interrupts Definition

ADC_IT_RDY	ADC Ready (ADRDY) interrupt source
ADC_IT_EOSMP	ADC End of Sampling interrupt source
ADC_IT_EOC	ADC End of Regular Conversion interrupt source
ADC_IT_EOS	ADC End of Regular sequence of Conversions interrupt source
ADC_IT_OVR	ADC overrun interrupt source
ADC_IT_JEOC	ADC End of Injected Conversion interrupt source
ADC_IT_JEOS	ADC End of Injected sequence of Conversions interrupt source
ADC_IT_AWD1	ADC Analog watchdog 1 interrupt source (main analog watchdog, present on all STM32 devices)

ADC_IT_AWD2	ADC Analog watchdog 2 interrupt source (additional analog watchdog, present only on STM32F3 devices)
ADC_IT_AWD3	ADC Analog watchdog 3 interrupt source (additional analog watchdog, present only on STM32F3 devices)
ADC_IT_JQOVF	ADC Injected Context Queue Overflow interrupt source
ADC_IT_AWD	

ADC Extended Offset Number

ADC_OFFSET_NONE

ADC_OFFSET_1

ADC_OFFSET_2

ADC_OFFSET_3

ADC_OFFSET_4

ADC Extended overrun

ADC_OVR_DATA_OVERWRITTEN	Default setting, to be used for compatibility with other STM32 devices
ADC_OVR_DATA_PRESERVED	

ADC Extended Private Constants

ADC_CALIBRATION_TIMEOUT

ADC_ENABLE_TIMEOUT

ADC_DISABLE_TIMEOUT

ADC_STOP_CONVERSION_TIMEOUT

ADC_CONVERSION_TIME_MAX_CPU_CYCLES

ADC_STAB_DELAY_US

ADC_TEMPSENSOR_DELAY_US

ADCEx Private Private Macros

ADC_ENABLING_CONDITIONS	**Description:** • Verification of hardware constraints before ADC can be enabled. **Parameters:** • __HANDLE__: ADC handle **Return value:** • SET: (ADC can be enabled) or RESET (ADC cannot be enabled)
ADC_IS_ENABLED	**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_IS_CONVERSION_ONGOING_REGULAR_INJECTED	**Description:** • Check if no conversion on going on regular and/or injected groups. **Parameters:** • __HANDLE__: ADC handle **Return value:** • SET: (conversion is on going) or RESET (no conversion is on going)
ADC_IS_CONVERSION_ONGOING_REGULAR	**Description:** • Check if no conversion on going on regular group. **Parameters:** • __HANDLE__: ADC handle **Return value:** • SET: (conversion is on going) or RESET (no conversion is on going)

ADC_IS_CONVERSION_ONGOING_INJECTED	**Description:** • Check if no conversion on going on injected group. **Parameters:** • __HANDLE__: ADC handle **Return value:** • SET: (conversion is on going) or RESET (no conversion is on going)
ADC_GET_RESOLUTION	**Description:** • Returns resolution bits in CFGR1 register: RES[1:0]. **Parameters:** • __HANDLE__: ADC handle **Return value:** • None:
ADC_CLEAR_ERRORCODE	**Description:** • Clear ADC error code (set it to error code: "no error") **Parameters:** • __HANDLE__: ADC handle **Return value:** • None:
ADC_SMPR1	**Description:** • Set the ADC's sample time for Channels numbers between 0 and 9. **Parameters:** • _SAMPLETIME_: Sample time parameter. • _CHANNELNB_: Channel number. **Return value:** • None:
ADC_SMPR2	**Description:** • Set the ADC's sample time for Channels numbers between 10 and 18. **Parameters:** • _SAMPLETIME_: Sample time parameter.

- _CHANNELNB_: Channel number.

Return value:

- None:

ADC_SQR1_RK

Description:

- Set the selected regular Channel rank for rank between 1 and 4.

Parameters:

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

Return value:

- None:

ADC_SQR2_RK

Description:

- Set the selected regular Channel rank for rank between 5 and 9.

Parameters:

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

Return value:

- None:

ADC_SQR3_RK

Description:

- Set the selected regular Channel rank for rank between 10 and 14.

Parameters:

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

Return value:

- None:

ADC_SQR4_RK

Description:

- Set the selected regular Channel rank for rank between 15 and 16.

Parameters:

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

Return value:

- None:

ADC_JSQR_RK

Description:

- Set the selected injected Channel rank.

Parameters:

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

Return value:

- None:

ADC_CFGR_AWD1CH_SHIFT

Description:

- Set the Analog Watchdog 1 channel.

Parameters:

- _CHANNEL_: channel to be monitored by Analog Watchdog 1.

Return value:

- None:

ADC_CFGR_AWD23CR

Description:

- Configure the channel number into Analog Watchdog 2 or 3.

Parameters:

- _CHANNEL_: ADC Channel

Return value:

- None:

ADC_CFGR_INJECT_AUTO_CONVERSION

Description:

- Enable automatic conversion of injected group.

Parameters:

- _INJECT_AUTO_CONVERSION_: Injected automatic conversion.

Return value:

- None:

ADC_CFGR_INJECT_CONTEXT_QUEUE

Description:

- Enable ADC injected context queue.

Parameters:

- _INJECT_CONTEXT_QUEUE_

MODE_: Injected context queue mode.

Return value:

- None:

ADC_CFGR_INJECT_DISCCONTINUOUS

Description:

- Enable ADC discontinuous conversion mode for injected group.

Parameters:

- _INJECT_DISCONTINUOUS_MODE_: Injected discontinuous mode.

Return value:

- None:

ADC_CFGR_REG_DISCCONTINUOUS

Description:

- Enable ADC discontinuous conversion mode for regular group.

Parameters:

- _REG_DISCONTINUOUS_MODE_: Regular discontinuous mode.

Return value:

- None:

ADC_CFGR_DISCONTINUOUS_NUM

Description:

- Configures the number of discontinuous conversions for regular group.

Parameters:

- _NBR_DISCONTINUOUS_CONV_: Number of discontinuous conversions.

Return value:

- None:

ADC_CFGR_AUTOWAIT

Description:

- Enable the ADC auto delay mode.

Parameters:

- _AUTOWAIT_: Auto delay bit enable or disable.

Return value:

- None:

ADC_CFGR_CONTINUOUS

Description:

- Enable ADC continuous conversion mode.

Parameters:

- _CONTINUOUS_MODE_: Continuous mode.

Return value:

- None:

ADC_CFGR_OVERRUN

Description:

- Enable ADC overrun mode.

Parameters:

- _OVERRUN_MODE_: Overrun mode.

Return value:

- Overrun: bit setting to be programmed into CFGR register

ADC_CFGR_DMACONTREQ

Description:

- Enable the ADC DMA continuous request.

Parameters:

- _DMACONTREQ_MODE_: DMA continuous request mode.

Return value:

- None:

ADC_CFGR_EXTSEL_SET

Description:

- For devices with 3 ADCs or more: Defines the external trigger source for regular group according to ADC into common group ADC1&ADC2 or ADC3&ADC4 (some triggers with same source have different value to be programmed into ADC EXTSEL bits of CFGR 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 CFGR register

ADC_JSQR_JEXTSEL_SET

Description:

- For devices with 3 ADCs or more: Defines the external trigger source for injected group according to ADC into common group ADC1&ADC2 or ADC3&ADC4 (some triggers with same source have different value to be programmed into ADC JEXTSEL bits of JSQR 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 JSQR register

ADC_OFR_CHANNEL

Description:

- Configure the channel number into offset OFRx register.

Parameters:

- _CHANNEL_: ADC Channel

Return value:

- None:

ADC_DIFSEL_CHANNEL

Description:

- Configure the channel number into differential mode selection register.

Parameters:

- _CHANNEL_: ADC Channel

Return value:

- None:

ADC_CALFACT_DIFF_SET

Description:

- Calibration factor in differential mode to be set into calibration register.

Parameters:

	• _Calibration_Factor_: Calibration factor value **Return value:** • None:
ADC_CALFACT_DIFF_GET	**Description:** • Calibration factor in differential mode to be retrieved from calibration register. **Parameters:** • _Calibration_Factor_: Calibration factor value **Return value:** • None:
ADC_TRX_HIGHTHRESHOLD	**Description:** • Configure the analog watchdog high threshold into registers TR1, TR2 or TR3. **Parameters:** • _Threshold_: Threshold value **Return value:** • None:
ADC_CCR_MULTI_DMACONTREQ	**Description:** • Enable the ADC DMA continuous request for ADC multimode. **Parameters:** • _DMAContReq_MODE_: DMA continuous request mode. **Return value:** • None:
ADC_DISABLING_CONDITIONS	**Description:** • Verification of hardware constraints before ADC can be disabled. **Parameters:** • __HANDLE__: ADC handle **Return value:** • SET: (ADC can be disabled) or RESET (ADC cannot be disabled)

ADC_OFFSET_SHIFT_RESOLUTION	**Description:** • Shift the offset in function of the selected ADC resolution. **Parameters:** • __HANDLE__: ADC handle • _Offset_: Value to be shifted **Return value:** • None:
ADC_AWD1THRESHOLD_SHIFT_RESOLUTION	**Description:** • Shift the AWD1 threshold in function of the selected ADC resolution. **Parameters:** • __HANDLE__: ADC handle • _Threshold_: Value to be shifted **Return value:** • None:
ADC_AWD23THRESHOLD_SHIFT_RESOLUTION	**Description:** • Shift the AWD2 and AWD3 threshold in function of the selected ADC resolution. **Parameters:** • __HANDLE__: ADC handle • _Threshold_: Value to be shifted **Return value:** • None:
ADC_COMMON_REGISTER	**Description:** • Defines if the selected ADC is within ADC common register ADC1_2 or ADC3_4 if available (ADC2, ADC3, ADC4 availability depends on STM32 product) **Parameters:** • __HANDLE__: ADC handle **Return value:** • Common: control register ADC1_2 or ADC3_4
ADC_COMMON_CCR_MULTI	**Description:** • Selection of ADC common register CCR bits MULTI[4:0]corresponding to the selected ADC (applicable for

devices with several ADCs)

Parameters:

- __HANDLE__: ADC handle

Return value:

- None:

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 same common register ADC1_2 or ADC3_4 if available (ADC2, ADC3, ADC4 availability depends on STM32 product)

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 if available (ADC2, ADC3, ADC4 availability depends on STM32 product)

Parameters:

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

Return value:

- None:

IS_ADC_RESOLUTION

IS_ADC_RESOLUTION_8_6_BITS

IS_ADC_DATA_ALIGN

IS_ADC_SCAN_MODE

IS_ADC_EOC_SELECTION

IS_ADC_OVERRUN

IS_ADC_CHANNEL

IS_ADC_DIFF_CHANNEL

IS_ADC_SAMPLE_TIME

IS_ADC_SINGLE_DIFFERENTIAL

IS_ADC_OFFSET_NUMBER

IS_ADC_REGULAR_RANK

IS_ADC_EXTTRIG_EDGE

IS_ADC_EXTTRIG

IS_ADC_EXTTRIGINJEC_EDGE

IS_ADC_EXTTRIGINJEC

IS_ADC_INJECTED_RANK

IS_ADC_MODE

IS_ADC_DMA_ACCESS_MODE

IS_ADC_SAMPLING_DELAY

IS_ADC_ANALOG_WATCHDOG_NUMBER

IS_ADC_ANALOG_WATCHDOG_MODE

IS_ADC_CONVERSION_GROUP

IS_ADC_EVENT_TYPE

IS_ADC_IT

IS_ADC_FLAG

ADC Extended Range Verification

IS_ADC_RANGE

ADC Extended 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 Extended Resolution

ADC_RESOLUTION12B	ADC 12-bit resolution
ADC_RESOLUTION10B	ADC 10-bit resolution
ADC_RESOLUTION8B	ADC 8-bit resolution
ADC_RESOLUTION6B	ADC 6-bit resolution

ADC Extended Sampling Times

ADC_SAMPLETIME_1CYCLE_5	Sampling time 1.5 ADC clock cycle
ADC_SAMPLETIME_2CYCLES_5	Sampling time 2.5 ADC clock cycles
ADC_SAMPLETIME_4CYCLES_5	Sampling time 4.5 ADC clock cycles
ADC_SAMPLETIME_7CYCLES_5	Sampling time 7.5 ADC clock cycles
ADC_SAMPLETIME_19CYCLES_5	Sampling time 19.5 ADC clock cycles
ADC_SAMPLETIME_61CYCLES_5	Sampling time 61.5 ADC clock cycles
ADC_SAMPLETIME_181CYCLES_5	Sampling time 181.5 ADC clock cycles
ADC_SAMPLETIME_601CYCLES_5	Sampling time 601.5 ADC clock cycles

ADC Extended Scan Mode

ADC_SCAN_DISABLE

ADC_SCAN_ENABLE

ADC Extended Single-ended/Differential input mode

ADC_SINGLE_ENDED

ADC_DIFFERENTIAL_ENDED

6 HAL CAN Generic Driver

6.1 CAN Firmware driver registers structures

6.1.1 CAN_InitTypeDef

CAN_InitTypeDef is defined in the stm32f3xx_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 CAN_FilterConfTypeDef

CAN_FilterConfTypeDef is defined in the stm32f3xx_hal_can.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 = 27.

- ***uint32_t CAN_FilterConfTypeDef::FilterMode***
  Specifies the filter mode to be initialized. This parameter can be a value of ***CAN_filter_mode***
- ***uint32_t CAN_FilterConfTypeDef::FilterScale***
  Specifies the filter scale. This parameter can be a value of ***CAN_filter_scale***
- ***uint32_t CAN_FilterConfTypeDef::FilterActivation***
  Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t CAN_FilterConfTypeDef::BankNumber***
  Select the start slave bank filter This parameter must be a number between Min_Data = 0 and Max_Data = 28.

6.1.3 CanTxMsgTypeDef

CanTxMsgTypeDef is defined in the stm32f3xx_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.

6.1.4 CanRxMsgTypeDef

CanRxMsgTypeDef is defined in the stm32f3xx_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 CAN_FIFO0 or CAN_FIFO1

6.1.5 CAN_HandleTypeDef

CAN_HandleTypeDef is defined in the stm32f3xx_hal_can.h

Data Fields

- ***CAN_TypeDef * Instance***
- ***CAN_InitTypeDef Init***
- ***CanTxMsgTypeDef * pTxMsg***
- ***CanRxMsgTypeDef * pRxMsg***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CAN_StateTypeDef State***
- ***__IO HAL_CAN_ErrorTypeDef 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 HAL_CAN_ErrorTypeDef 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 __CAN_CLK_ENABLE();
2. CAN pins configuration
   - Enable the clock for the CAN GPIOs using the following function: __GPIOx_CLK_ENABLE();
   - Connect and configure the involved CAN pins to AF9 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:** Timeout duration.
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. • **Timeout:** Timeout duration.
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:** 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 Clock Prescaler

IS_CAN_PRESCALER

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

Return value:
- None:

__HAL_CAN_DISABLE_IT

Description:
- Disable the specified CAN interrupts.

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

Return value:
- None:

__HAL_CAN_MSG_PENDING

Description:
- Return the number of pending received messages.

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

Return value:

- The: number of pending message.

CAN_FLAG_MASK

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_GET_FLAG

__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
  - 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_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: FIFO0 message pending interrupt enablev
  - CAN_IT_FMP1: FIFO1 message pending interrupt enable

Return value:

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

__HAL_CAN_TRANSMIT_STATUS

Description:

- Check the transmission status of a CAN Frame.

Parameters:

- __HANDLE__: 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
IS_CAN_FILTER_FIFO	
CAN_FilterFIFO0	
CAN_FilterFIFO1	

CAN Filter Mode

CAN_FILTERMODE_IDMASK	Identifier mask mode
CAN_FILTERMODE_IDLIST	Identifier list mode
IS_CAN_FILTER_MODE	

CAN Filter Number

IS_CAN_FILTER_NUMBER

CAN Filter Scale

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

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
IS_CAN_GET_FLAG	
IS_CAN_CLEAR_FLAG	

CAN Identifier Type

CAN_ID_STD	Standard Id
CAN_ID_EXT	Extended Id
IS_CAN_IDTYPE	

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_IT_RQCP0	
CAN_IT_RQCP1	
CAN_IT_RQCP2	
IS_CAN_IT	

IS_CAN_CLEAR_IT

CAN Mailboxes

CAN_TXMAILBOX_0

CAN_TXMAILBOX_1

CAN_TXMAILBOX_2

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

IS_CAN_MODE

CAN Private Constants

HAL_CAN_DEFAULT_TIMEOUT

CAN Receive FIFO Number

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

IS_CAN_FIFO

CAN Remote Transmission Request

CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame

IS_CAN_RTR

CAN Start Bank Filter For Slave CAN

IS_CAN_BANKNUMBER

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

IS_CAN_SJW

CAN Timeouts

INAK_TIMEOUT

SLAK_TIMEOUT

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
IS_CAN_BS1	

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
IS_CAN_BS2	

CAN Transmit Constants

CAN_TXSTATUS_FAILED	CAN transmission failed
CAN_TXSTATUS_OK	CAN transmission succeeded
CAN_TXSTATUS_PENDING	CAN transmission pending
CAN_TXSTATUS_NOMAILBOX	CAN cell did not provide CAN_TxStatus_NoMailBox

CAN Tx

IS_CAN_TRANSMITMAILBOX

IS_CAN_STDID

IS_CAN_EXTID

IS_CAN_DLC

7 HAL CEC Generic Driver

7.1 CEC Firmware driver registers structures

7.1.1 CEC_InitTypeDef

CEC_InitTypeDef is defined in the stm32f3xx_hal_cec.h

Data Fields

- ***uint32_t SignalFreeTime***
- ***uint32_t Tolerance***
- ***uint32_t BRERxStop***
- ***uint32_t BREErrorBitGen***
- ***uint32_t LBPEErrorBitGen***
- ***uint32_t BroadcastMsgNoErrorBitGen***
- ***uint32_t SignalFreeTimeOption***
- ***uint32_t OwnAddress***
- ***uint32_t ListenMode***
- ***uint8_t InitiatorAddress***

Field Documentation

- ***uint32_t CEC_InitTypeDef::SignalFreeTime***
  Set SFT field, specifies the Signal Free Time. It can be one of ***CEC_Signal_Free_Time*** and belongs to the set {0,...,7} where 0x0 is the default configuration else means 0.5 + (SignalFreeTime - 1) nominal data bit periods
- ***uint32_t CEC_InitTypeDef::Tolerance***
  Set RXTOL bit, specifies the tolerance accepted on the received waveforms, it can be a value of ***CEC_Tolerance*** : it is either CEC_STANDARD_TOLERANCE or CEC_EXTENDED_TOLERANCE
- ***uint32_t CEC_InitTypeDef::BRERxStop***
  Set BRESTP bit ***CEC_BRERxStop*** : specifies whether or not a Bit Rising Error stops the reception. CEC_NO_RX_STOP_ON_BRE: reception is not stopped. CEC_RX_STOP_ON_BRE: reception is stopped.
- ***uint32_t CEC_InitTypeDef::BREErrorBitGen***
  Set BREGEN bit ***CEC_BREErrorBitGen*** : specifies whether or not an Error-Bit is generated on the CEC line upon Bit Rising Error detection. CEC_BRE_ERRORBIT_NO_GENERATION: no error-bit generation. CEC_BRE_ERRORBIT_GENERATION: error-bit generation if BRESTP is set.
- ***uint32_t CEC_InitTypeDef::LBPEErrorBitGen***
  Set LBPEGEN bit ***CEC_LBPEErrorBitGen*** : specifies whether or not an Error-Bit is generated on the CEC line upon Long Bit Period Error detection. CEC_LBPE_ERRORBIT_NO_GENERATION: no error-bit generation. CEC_LBPE_ERRORBIT_GENERATION: error-bit generation.
- ***uint32_t CEC_InitTypeDef::BroadcastMsgNoErrorBitGen***
  Set BRDNOGEN bit ***CEC_BroadCastMsgErrorBitGen*** : allows to avoid an Error-Bit generation on the CEC line upon an error detected on a broadcast message.It supersedes BREGEN and LBPEGEN bits for a broadcast message error handling. It can take two values:1) CEC_BROADCASTERROR_ERRORBIT_GENERATION. a) BRE detection: error-bit generation on the CEC line if

BRESTP=CEC_RX_STOP_ON_BRE and BREGEN=CEC_BRE_ERRORBIT_NO_GENERATION. b) LBPE detection: error-bit generation on the CEC line if LBPGEN=CEC_LBPE_ERRORBIT_NO_GENERATION.2) CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION. no error-bit generation in case neither a) nor b) are satisfied. Additionally, there is no error-bit generation in case of Short Bit Period Error detection in a broadcast message while LSTN bit is set.

- ***uint32_t CEC_InitTypeDef::SignalFreeTimeOption***
  Set SFTOP bit ***CEC_SFT_Option*** : specifies when SFT timer starts. CEC_SFT_START_ON_TXSOM SFT: timer starts when TXSOM is set by software. CEC_SFT_START_ON_TX_RX_END: SFT timer starts automatically at the end of message transmission/reception.
- ***uint32_t CEC_InitTypeDef::OwnAddress***
  Set OAR field, specifies CEC device address within a 15-bit long field
- ***uint32_t CEC_InitTypeDef::ListenMode***
  Set LSTN bit ***CEC_Listening_Mode*** : specifies device listening mode. It can take two values:CEC_REDUCED_LISTENING_MODE: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.CEC_FULL_LISTENING_MODE: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.
- ***uint8_t CEC_InitTypeDef::InitiatorAddress***

7.1.2 CEC_HandleTypeDef

CEC_HandleTypeDef is defined in the stm32f3xx_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_InitTypeDef CEC_HandleTypeDef::Init***
- ***uint8_t* CEC_HandleTypeDef::pTxBuffPtr***
- ***uint16_t CEC_HandleTypeDef::TxXferCount***
- ***uint8_t* CEC_HandleTypeDef::pRxBuffPtr***
- ***uint16_t CEC_HandleTypeDef::RxXferSize***
- ***uint32_t CEC_HandleTypeDef::ErrorCode***
- ***HAL_LockTypeDef CEC_HandleTypeDef::Lock***
- ***HAL_CEC_StateTypeDef CEC_HandleTypeDef::State***

7.2 CEC Firmware driver API description

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

7.2.1 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:
   - Enable the CEC interface clock.
   - CEC pins configuration:
     - Enable the clock for the CEC GPIOs.
     - Configure these CEC pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_CEC_Transmit_IT() and HAL_CEC_Receive_IT() APIs):
     - Configure the CEC interrupt priority.
     - Enable the NVIC CEC IRQ handle.
3. Program the Signal Free Time (SFT) and SFT option, Tolerance, reception stop in in case of Bit Rising Error, Error-Bit generation conditions, device logical address and Listen mode in the hcec Init structure.
4. Initialize the CEC registers by calling the HAL_CEC_Init() API.
   - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customed HAL_CEC_MspInit() API. The specific CEC interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_CEC_ENABLE_IT() and __HAL_CEC_DISABLE_IT() inside the transmit and receive process.

7.2.2 Initialization and Configuration functions

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

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

7.2.3 IO operation function

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

7.2.4 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_GetState()*
- *HAL_CEC_GetError()*

7.2.5 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

7.2.6 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

7.2.7 HAL_CEC_MspInit

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

7.2.8 HAL_CEC_MspDeInit

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

7.2.9 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

7.2.10 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. Note that 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
Return values	• HAL status

7.2.11 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

7.2.12 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. Note that 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
Return values	• HAL status

7.2.13 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

7.2.14 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

7.2.15 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

7.2.16 HAL_CEC_ErrorCallback

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

7.2.17 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

7.2.18 HAL_CEC_GetError

Function Name	**uint32_t HAL_CEC_GetError (CEC_HandleTypeDef * hcec)**
Function Description	Return the CEC error code.
Parameters	• **hcec:** : pointer to a CEC_HandleTypeDef structure that contains the configuration information for the specified CEC.
Return values	• CEC Error Code

7.3 CEC Firmware driver defines

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

7.3.1 CEC

CEC

all RX or TX errors flags in CEC ISR register

CEC_ISR_ALL_ERROR

Error Bit Generation if Bit Rise Error reported

CEC_BRE_ERRORBIT_NO_GENERATION

CEC_BRE_ERRORBIT_GENERATION

IS_CEC_BREERRORBITGEN

Reception Stop on Error

CEC_NO_RX_STOP_ON_BRE

CEC_RX_STOP_ON_BRE

IS_CEC_BRERXSTOP

Error Bit Generation on Broadcast message

CEC_BROADCASTERROR_ERRORBIT_GENERATION

CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION

IS_CEC_BROADCASTERROR_NO_ERRORBIT_GENERATION

CEC Exported Macros

__HAL_CEC_RESET_HANDLE_STATE

Description:

- Reset CEC handle state.

Parameters:

- __HANDLE__: CEC handle.

Return value:

- None:

__HAL_CEC_GET_IT

Description:

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

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the interrupt to check. This parameter can be one of the following values:
  - CEC_ISR_RXBR : Rx-Byte Received
  - CEC_ISR_RXEND : End of Reception
  - CEC_ISR_RXOVR : Rx Overrun
  - CEC_ISR_BRE : Rx Bit Rising Error
  - CEC_ISR_SBPE : Rx Short Bit Period Error
  - CEC_ISR_LBPE : Rx Long Bit Period Error
  - CEC_ISR_RXACKE : Rx Missing Acknowledge
  - CEC_ISR_ARBLST : Arbitration lost
  - CEC_ISR_TXBR : Tx-Byte Request
  - CEC_ISR_TXEND : End of Transmission
  - CEC_ISR_TXUDR : Tx-buffer Underrun
  - CEC_ISR_TXERR : Tx Error
  - CEC_ISR_TXACKE : Tx

Missing Acknowledge

Return value:

- ITStatus:

__HAL_CEC_CLEAR_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the interrupt/status flag to clear. This parameter can be one of the following values:
  - CEC_ISR_RXBR : Rx-Byte Received
  - CEC_ISR_RXEND : End of Reception
  - CEC_ISR_RXOVR : Rx Overrun
  - CEC_ISR_BRE : Rx Bit Rising Error
  - CEC_ISR_SBPE : Rx Short Bit Period Error
  - CEC_ISR_LBPE : Rx Long Bit Period Error
  - CEC_ISR_RXACKE : Rx Missing Acknowledge
  - CEC_ISR_ARBLST : Arbitration lost
  - CEC_ISR_TXBR : Tx-Byte Request
  - CEC_ISR_TXEND : End of Transmission
  - CEC_ISR_TXUDR : Tx-buffer Underrun
  - CEC_ISR_TXERR : Tx Error
  - CEC_ISR_TXACKE : Tx Missing Acknowledge

Return value:

- none:

__HAL_CEC_ENABLE_IT

Description:

- Enables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies

the CEC interrupt to enable. This parameter can be one of the following values:
  - CEC_IER_RXBRIE : Rx-Byte Received IT Enable
  - CEC_IER_RXENDIE : End Of Reception IT Enable
  - CEC_IER_RXOVRIE : Rx-Overrun IT Enable
  - CEC_IER_BREIE : Rx Bit Rising Error IT Enable
  - CEC_IER_SBPEIE : Rx Short Bit period Error IT Enable
  - CEC_IER_LBPEIE : Rx Long Bit period Error IT Enable
  - CEC_IER_RXACKEIE : Rx Missing Acknowledge IT Enable
  - CEC_IER_ARBLSTIE : Arbitration Lost IT Enable
  - CEC_IER_TXBRIE : Tx Byte Request IT Enable
  - CEC_IER_TXENDIE : End of Transmission IT Enable
  - CEC_IER_TXUDRIE : Tx-Buffer Underrun IT Enable
  - CEC_IER_TXERRIE : Tx-Error IT Enable
  - CEC_IER_TXACKEIE : Tx Missing Acknowledge IT Enable

Return value:

- none:

__HAL_CEC_DISABLE_IT

Description:

- Disables the specified CEC interrupt.

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to disable. This parameter can be one of the following values:
  - CEC_IER_RXBRIE : Rx-Byte Received IT Enable
  - CEC_IER_RXENDIE : End Of Reception IT Enable

- CEC_IER_RXOVRIE : Rx-Overrun IT Enable
- CEC_IER_BREIE : Rx Bit Rising Error IT Enable
- CEC_IER_SBPEIE : Rx Short Bit period Error IT Enable
- CEC_IER_LBPEIE : Rx Long Bit period Error IT Enable
- CEC_IER_RXACKEIE : Rx Missing Acknowledge IT Enable
- CEC_IER_ARBLSTIE : Arbitration Lost IT Enable
- CEC_IER_TXBRIE : Tx Byte Request IT Enable
- CEC_IER_TXENDIE : End of Transmission IT Enable
- CEC_IER_TXUDRIE : Tx-Buffer Underrun IT Enable
- CEC_IER_TXERRIE : Tx-Error IT Enable
- CEC_IER_TXACKEIE : Tx Missing Acknowledge IT Enable

Return value:

- none:

__HAL_CEC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __INTERRUPT__: specifies the CEC interrupt to check. This parameter can be one of the following values:
  - CEC_IER_RXBRIE : Rx-Byte Received IT Enable
  - CEC_IER_RXENDIE : End Of Reception IT Enable
  - CEC_IER_RXOVRIE : Rx-Overrun IT Enable
  - CEC_IER_BREIE : Rx Bit Rising Error IT Enable
  - CEC_IER_SBPEIE : Rx Short Bit period Error IT Enable

- CEC_IER_LBPEIE : Rx Long Bit period Error IT Enable
- CEC_IER_RXACKEIE : Rx Missing Acknowledge IT Enable
- CEC_IER_ARBLSTIE : Arbitration Lost IT Enable
- CEC_IER_TXBRIE : Tx Byte Request IT Enable
- CEC_IER_TXENDIE : End of Transmission IT Enable
- CEC_IER_TXUDRIE : Tx-Buffer Underrun IT Enable
- CEC_IER_TXERRIE : Tx-Error IT Enable
- CEC_IER_TXACKEIE : Tx Missing Acknowledge IT Enable

Return value:

- FlagStatus:

__HAL_CEC_ENABLE

Description:

- Enables the CEC device.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none:

__HAL_CEC_DISABLE

Description:

- Disables the CEC device.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none:

__HAL_CEC_FIRST_BYTE_TX_SET

Description:

- Set Transmission Start flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

- none:

__HAL_CEC_LAST_BYTE_TX_SET

Description:

- Set Transmission End flag.

Parameters:

- __HANDLE__: specifies the CEC Handle.

Return value:

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

__HAL_CEC_GET_TRANSMISSION_START_FLAG	**Description:** • Get Transmission Start flag. **Parameters:** • __HANDLE__: specifies the CEC Handle. **Return value:** • FlagStatus:
__HAL_CEC_GET_TRANSMISSION_END_FLAG	**Description:** • Get Transmission End flag. **Parameters:** • __HANDLE__: specifies the CEC Handle. **Return value:** • FlagStatus:
__HAL_CEC_CLEAR_OAR	**Description:** • Clear OAR register. **Parameters:** • __HANDLE__: specifies the CEC Handle. **Return value:** • none:
__HAL_CEC_SET_OAR	**Description:** • Set OAR register (without resetting previously set address in case of multi-address mode) To reset OAR, **Parameters:** • __HANDLE__: specifies the CEC Handle. • __ADDRESS__: Own Address value (CEC logical address is identified by bit position)

Return value:

- none:

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).

all RX errors interrupts enabling flag

CEC_IER_RX_ALL_ERR

all TX errors interrupts enabling flag

CEC_IER_TX_ALL_ERR

Initiator logical address position in message header

CEC_INITIATOR_LSB_POS

Error Bit Generation if Long Bit Period Error reported

CEC_LBPE_ERRORBIT_NO_GENERATION

CEC_LBPE_ERRORBIT_GENERATION

IS_CEC_LBPEERRORBITGEN

Listening mode option

CEC_REDUCED_LISTENING_MODE

CEC_FULL_LISTENING_MODE

IS_CEC_LISTENING_MODE

Device Own Address position in CEC CFGR register

CEC_CFGR_OAR_LSB_POS

CEC Private Constants

CEC_CFGR_FIELDS

Signal Free Time start option

CEC_SFT_START_ON_TXSOM

CEC_SFT_START_ON_TX_RX_END

IS_CEC_SFTOP

Signal Free Time setting parameter

CEC_DEFAULT_SFT

CEC_0_5_BITPERIOD_SFT

CEC_1_5_BITPERIOD_SFT

CEC_2_5_BITPERIOD_SFT

CEC_3_5_BITPERIOD_SFT

CEC_4_5_BITPERIOD_SFT

CEC_5_5_BITPERIOD_SFT

CEC_6_5_BITPERIOD_SFT

IS_CEC_SIGNALFREETIME

Receiver Tolerance

CEC_STANDARD_TOLERANCE

CEC_EXTENDED_TOLERANCE

IS_CEC_TOLERANCE

8 HAL COMP Generic Driver

8.1 COMP Firmware driver registers structures

8.1.1 COMP_InitTypeDef

COMP_InitTypeDef is defined in the stm32f3xx_hal_comp.h

Data Fields

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

Field Documentation

- ***uint32_t COMP_InitTypeDef::InvertingInput***
  Selects the inverting input of the comparator. This parameter can be a value of ***COMPEx_InvertingInput***
- ***uint32_t COMP_InitTypeDef::NonInvertingInput***
  Selects the non inverting input of the comparator. This parameter can be a value of ***COMPEx_NonInvertingInput*** Note: Only available on STM32F302xB/xC, STM32F303xB/xC and STM32F358xx devices
- ***uint32_t COMP_InitTypeDef::Output***
  Selects the output redirection of the comparator. This parameter can be a value of ***COMPEx_Output***
- ***uint32_t COMP_InitTypeDef::OutputPol***
  Selects the output polarity of the comparator. This parameter can be a value of ***COMP_OutputPolarity***
- ***uint32_t COMP_InitTypeDef::Hysteresis***
  Selects the hysteresis voltage of the comparator. This parameter can be a value of ***COMPEx_Hysteresis*** Note: Only available on STM32F302xB/xC, STM32F303xB/xC, STM32F373xB/xC, STM32F358xx and STM32F378xx devices
- ***uint32_t COMP_InitTypeDef::BlankingSrce***
  Selects the output blanking source of the comparator. This parameter can be a value of ***COMPEx_BlankingSrce*** Note: Not available on STM32F373xB/xC and STM32F378xx devices
- ***uint32_t COMP_InitTypeDef::Mode***
  Selects the operating comsumption mode of the comparator to adjust the speed/consumption. This parameter can be a value of ***COMPEx_Mode*** Note: Not available on STM32F301x6/x8, STM32F302x6/x8, STM32F334x6/x8, STM32F318xx and STM32F328xx devices
- ***uint32_t COMP_InitTypeDef::WindowMode***
  Selects the window mode of the comparator X (X=2, 4 or 6 if available). This parameter can be a value of ***COMPEx_WindowMode***

- ***uint32_t COMP_InitTypeDef::TriggerMode***
  Selects the trigger mode of the comparator (interrupt mode). This parameter can be a value of ***COMP_TriggerMode***

8.1.2 COMP_HandleTypeDef

COMP_HandleTypeDef is defined in the stm32f3xx_hal_comp.h

Data Fields

- ***COMP_TypeDef * Instance***
- ***COMP_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_COMP_StateTypeDef State***

Field Documentation

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

8.2 COMP Firmware driver API description

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

8.2.1 COMP Peripheral features

The STM32F3xx device family integrates up to 7 analog comparators COMP1, COMP2...COMP7:

1. The non inverting input and inverting input can be set to GPIO pins as shown in *Table 16: "COMP Inputs for STM32F303xB/STM32F303xC/STM32F303xE devices"*. COMP Inputs below for STM32F303xB/STM32F303xC as example. For other STM32F3xx devices please refer to the COMP peripheral section in corresponding Reference Manual.
2. The COMP output is available using HAL_COMP_GetOutputLevel() and can be set on GPIO pins (refer to *Table 17: "COMP outputs for STM32F303xB/STM32F303xC/STM32F303xE devices"*). COMP Outputs below for STM32F303xB/STM32F303xC as example. For other STM32F3xx devices please refer to the COMP peripheral section in corresponding Reference Manual.
3. The COMP output can be redirected to embedded timers (TIM1, TIM2, TIM3...) (refer to *Table 18: "Redirection of COMP outputs to embedded timers for STM32F303xB/STM32F303xC devices"* and *Table 19: "Redirection of COMP outputs to embedded timers for STM32F303xE devices"*). COMP Outputs redirection to embedded timers below for STM32F303xB/STM32F303xC as example. For other

STM32F3xx devices please refer to the COMP peripheral section in corresponding Reference Manual.

4. The comparators COMP1 and COMP2, COMP3 and COMP4, COMP5 and COMP6 can be combined in window mode and only COMP1, COMP3 and COMP5 non inverting input can be used as non-inverting input.
5. The seven comparators have interrupt capability with wake-up from Sleep and Stop modes (through the EXTI controller):
   - COMP1 is internally connected to EXTI Line 21
   - COMP2 is internally connected to EXTI Line 22
   - COMP3 is internally connected to EXTI Line 29
   - COMP4 is internally connected to EXTI Line 30
   - COMP5 is internally connected to EXTI Line 31
   - COMP6 is internally connected to EXTI Line 32
   - COMP7 is internally connected to EXTI Line 33 From the corresponding IRQ handler, the right interrupt source can be retrieved with the macro __HAL_COMP_EXTI_GET_FLAG(). Possible values are:
   - COMP_EXTI_LINE_COMP1_EVENT
   - COMP_EXTI_LINE_COMP2_EVENT
   - COMP_EXTI_LINE_COMP3_EVENT
   - COMP_EXTI_LINE_COMP4_EVENT
   - COMP_EXTI_LINE_COMP5_EVENT
   - COMP_EXTI_LINE_COMP6_EVENT
   - COMP_EXTI_LINE_COMP7_EVENT

Table 16: COMP Inputs for STM32F303xB/STM32F303xC/STM32F303xE devices

		COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
Inverting inputs	1/4 VREFINT	OK	OK	OK	OK	OK	OK	OK
	1/2 VREFINT	OK	OK	OK	OK	OK	OK	OK
	3/4 VREFINT	OK	OK	OK	OK	OK	OK	OK
	VREFINT	OK	OK	OK	OK	OK	OK	OK
	DAC1 OUT (PA4)	OK	OK	OK	OK	OK	OK	OK
	DAC2 OUT (PA5)	OK	OK	OK	OK	OK	OK	OK
	I/O1	PA0	PA2	PD15	PE8	PD13	PD10	PC0
	I/O2	---	---	PB12	PB2	PB10	PB15	---
Non-inverting inputs	I/O1	PA1	PA7	PB14	PB0	PD12	PD11	PA0
	I/O2	---	PA3	PD14	PE7	PB13	PAB11	PC1

Table 17: COMP outputs for STM32F303xB/STM32F303xC/STM32F303xE devices

COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
PA0	PA2	PB1	PC8	PC7	PA10	PC2
PF4	PA7	---	PA8	PA9	PC6	---
PA6	PA12	---	---	---	---	---
PA11	PB9	---	---	---	---	---
PB8	---	---	---	---	---	---

Table 18: Redirection of COMP outputs to embedded timers for STM32F303xB/STM32F303xC devices

COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN
TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2
TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN
TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2
TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2
TIM1 OCREFCLR	TIM1 OCREFCLR	TIM1 OCREFCLR	TIM8 OCREFCLR	TIM8 OCREFCLR	TIM8 OCREFCLR	TIM1 OCREFCLR
TIM1 I2C1	TIM1 I2C1	TIM2 OCREFCLR	TIM3 I2C3	TIM2 I2C1	TIM2 I2C2	TIM8 OCREFCLR
TIM2 I2C4	TIM2 I2C4	TIM3 I2C2	TIM3 OCREFCLR	TIM3 OCREFCLR	TIM2 OCREFCLR	TIM2 I2C3
TIM2 OCREFCLR	TIM2 OCREFCLR	TIM4 I2C1	TIM4 I2C2	TIM4 I2C3	TIM16 OCREFCLR	TIM1 I2C2
TIM3 I2C1	TIM3 I2C1	TIM15 I2C1	TIM15 OCREFCLR	TIM16 BKIN	TIM16 I2C1	TIM17 OCREFCLR
TIM3 OCREFCLR	TIM3 OCREFCLR	TIM15 BKIN	TIM15 I2C2	TIM17 I2C1	TIM4 I2C4	TIM17 BKIN

Table 19: Redirection of COMP outputs to embedded timers for STM32F303xE devices

COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
TIM1 BKIN	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN(1)	TIM1 BKIN	TIM1 BKIN	TIM1 BKIN(1)
TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2	TIM1 BKIN2
TIM8 BKIN	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN(1)	TIM8 BKIN	TIM8 BKIN	TIM8 BKIN(1)
TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2	TIM8 BKIN2
TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2	TIM1 BKIN2 + TIM8 BKIN2

COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
TIM1 OCREFCLR	TIM1 OCREFCLR	TIM1 OCREFCLR	TIM8 OCREFCLR	TIM8 OCREFCLR	TIM8 OCREFCLR	TIM1 OCREFCLR
TIM1 I2C1	TIM1 I2C1	TIM2 OCREFCLR	TIM3 I2C3	TIM2 I2C1	TIM2 I2C2	TIM8 OCREFCLR
TIM2 I2C4	TIM2 I2C4	TIM3 I2C2	TIM3 OCREFCLR	TIM3 OCREFCLR	TIM2 OCREFCLR	TIM2 I2C3
TIM2 OCREFCLR	TIM2 OCREFCLR	TIM4 I2C1	TIM4 I2C2	TIM4 I2C3	TIM16 OCREFCLR	TIM1 I2C2
TIM3 I2C1	TIM3 I2C1	TIM15 I2C1	TIM15 OCREFCLR	TIM16 BKIN	TIM16 I2C1	TIM17 OCREFCLR
TIM3 OCREFCLR	TIM3 OCREFCLR	TIM15 BKIN	TIM15 I2C2	TIM17 I2C1	TIM4 I2C4	TIM17 BKIN
TIM20 BKIN	TIM20 BKIN	TIM20 BKIN	TIM20 BKIN(1)	TIM20 BKIN	TIM20 BKIN	TIM20 BKIN(1)
TIM20 BKIN2	TIM20 BKIN2	TIM20 BKIN2	TIM20 BKIN2	TIM20 BKIN2		TIM20 BKIN2
TIM1 BKIN2 + TIM8 BKIN2 + TIM20 BKIN2	TIM1 BKIN2 + TIM8 BKIN2 + TIM20 BKIN2	TIM1 BKIN2 + TIM8 BKIN2 + TIM20 BKIN2	TIM1 BKIN2 + TIM8 BKIN2 + TIM20 BKIN2	TIM1 BKIN2 + TIM8 BKIN2 + TIM20 BKIN2	TIM1 BKIN2 + TIM8 BKIN2 + TIM20 BKIN2	TIM1 BKIN2 + TIM8 BKIN2 + TIM20 BKIN2

Notes:

(1)This connection consists in connecting both GPIO and COMP outputs to TIM1/8/20 BRK input through an OR gate, instead of connecting the GPIO to TIM1/8/20 BRK input and the COMP output to TIM1/8/20 BRK_ACTH input. The objective is to add a 3-bit digital filter on the COMP output.

Table 20: COMP outputs blanking sources for the STM32F303xB/STM32F303xC/STM32F303xE devicesutputs for STM32F303xB/STM32F303xC/STM32F303xE devices

COMP1	COMP2	COMP3	COMP4	COMP5	COMP6	COMP7
TIM1 OC5	TIM1 OC5	TIM1 OC5	TIM3 OC4	---	TIM8 OC5	TIM1 OC5
TIM2 OC3	TIM2 OC3	---	TIM8 OC5	TIM3 OC3	TIM2 OC4	TIM8 OC5
TIM3 OC3	TIM3 OC3	TIM2 OC4	TIM15 OC1	TIM8 OC5	TIM15 OC2	TIM15 OC2

8.2.2 How to use this driver

This driver provides functions to configure and program the Comparators of all STM32F3xx devices. To use the comparator, perform the following steps:

1. Fill in the HAL_COMP_MspInit() to
   – Configure the comparator input in analog mode using HAL_GPIO_Init()

   - Configure the comparator output in alternate function mode using HAL_GPIO_Init() to map the comparator output to the GPIO pin
   - If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using HAL_GPIO_Init() function. After that enable the comparator interrupt vector using HAL_NVIC_EnableIRQ() function.
2. Configure the comparator using HAL_COMP_Init() function:
   - Select the inverting input
   - Select the non-inverting input
   - Select the output polarity
   - Select the output redirection
   - Select the hysteresis level
   - Select the power mode
   - Select the event/interrupt mode
3. Enable the comparator using HAL_COMP_Start() function or HAL_COMP_Start_IT() function for interrupt mode
4. Read the comparator output level with HAL_COMP_GetOutputLevel()

8.2.3 Initialization and Configuration functions

This section provides functions to initialize and de-initialize comparators

- ***HAL_COMP_Init()***
- ***HAL_COMP_DeInit()***
- ***HAL_COMP_MspInit()***
- ***HAL_COMP_MspDeInit()***

8.2.4 IO operation functions

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

- ***HAL_COMP_Start()***
- ***HAL_COMP_Stop()***
- ***HAL_COMP_Start_IT()***
- ***HAL_COMP_Stop_IT()***
- ***HAL_COMP_IRQHandler()***

8.2.5 Peripheral Control functions

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

- ***HAL_COMP_Lock()***
- ***HAL_COMP_GetOutputLevel()***
- ***HAL_COMP_TriggerCallback()***

8.2.6 Peripheral State functions

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

- ***HAL_COMP_GetState()***

8.2.7 HAL_COMP_Init

Function Name	**HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)**
Function Description	Initializes the COMP according to the specified parameters in the COMP_InitTypeDef and create the associated handle.
Parameters	• **hcomp:** COMP handle
Return values	• HAL status
Notes	• If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

8.2.8 HAL_COMP_DeInit

Function Name	**HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)**
Function Description	DeInitializes the COMP peripheral.
Parameters	• **hcomp:** COMP handle
Return values	• HAL status
Notes	• Deinitialization can't be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

8.2.9 HAL_COMP_MspInit

Function Name	**void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)**
Function Description	Initializes the COMP MSP.
Parameters	• **hcomp:** COMP handle
Return values	• None

8.2.10 HAL_COMP_MspDeInit

Function Name	**void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)**
Function Description	DeInitializes COMP MSP.
Parameters	• **hcomp:** COMP handle
Return values	• None

8.2.11 HAL_COMP_Start

Function Name	**HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)**
Function Description	Start the comparator.
Parameters	• **hcomp:** COMP handle
Return values	• HAL status

8.2.12 HAL_COMP_Stop

Function Name	**HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)**
Function Description	Stop the comparator.
Parameters	• **hcomp:** COMP handle
Return values	• HAL status

8.2.13 HAL_COMP_Start_IT

Function Name	**HAL_StatusTypeDef HAL_COMP_Start_IT (COMP_HandleTypeDef * hcomp)**
Function Description	Enables the interrupt and starts the comparator.
Parameters	• **hcomp:** COMP handle
Return values	• HAL status.

8.2.14 HAL_COMP_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_COMP_Stop_IT (COMP_HandleTypeDef * hcomp)**
Function Description	Disable the interrupt and Stop the comparator.
Parameters	• **hcomp:** COMP handle
Return values	• HAL status

8.2.15 HAL_COMP_IRQHandler

Function Name	**void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)**
Function Description	Comparator IRQ Handler.
Parameters	• **hcomp:** COMP handle
Return values	• HAL status

8.2.16 HAL_COMP_Lock

Function Name	**HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)**
Function Description	Lock the selected comparator configuration.
Parameters	• **hcomp:** COMP handle
Return values	• HAL status

8.2.17 HAL_COMP_GetOutputLevel

Function Name	**uint32_t HAL_COMP_GetOutputLevel (COMP_HandleTypeDef * hcomp)**
Function Description	Return the output level (high or low) of the selected comparator.

8.2.18 HAL_COMP_TriggerCallback

Function Name	**void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)**
Function Description	Comparator callback.
Parameters	• **hcomp:** COMP handle
Return values	• None

8.2.19 HAL_COMP_GetState

Function Name	**HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)**
Function Description	Return the COMP state.
Parameters	• **hcomp:** COMP handle
Return values	• HAL state

8.3 COMP Firmware driver defines

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

8.3.1 COMP

COMP

COMP Exported Constants

COMP_LOCK_DISABLE

COMP_LOCK_ENABLE

COMP_STATE_BIT_LOCK

COMP Exported Macros

__HAL_COMP_RESET_HANDLE_STATE	**Description:** • Reset COMP handle state. **Parameters:** • __HANDLE__: COMP handle. **Return value:** • None:

COMP Output Level

COMP_OUTPUTLEVEL_LOW

COMP_OUTPUTLEVEL_HIGH

COMP Output Polarity

COMP_OUTPUTPOL_NONINVERTED	COMP output on GPIO isn't inverted
COMP_OUTPUTPOL_INVERTED	COMP output on GPIO is inverted
IS_COMP_OUTPUTPOL	

COMP Trigger Mode

COMP_TRIGGERMODE_NONE	No External Interrupt trigger detection
COMP_TRIGGERMODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
COMP_TRIGGERMODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
COMP_TRIGGERMODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
IS_COMP_TRIGGERMODE	

9 HAL COMP Extension Driver

9.1 COMPEx Firmware driver defines

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

9.1.1 COMPEx

COMPEx

COMP Extended Blanking Source (STM32F303xE/STM32F398xx/STM32F303xC/STM32F358xx Product devices)

COMP_BLANKINGSRCE_NONE	No blanking source
COMP_BLANKINGSRCE_TIM1OC5	TIM1 OC5 selected as blanking source for comparator
COMP_BLANKINGSRCE_TIM2OC3	TIM2 OC5 selected as blanking source for comparator
COMP_BLANKINGSRCE_TIM3OC3	TIM2 OC3 selected as blanking source for comparator
COMP_BLANKINGSRCE_TIM2OC4	TIM2 OC4 selected as blanking source for comparator
COMP_BLANKINGSRCE_TIM8OC5	TIM8 OC5 selected as blanking source for comparator
COMP_BLANKINGSRCE_TIM3OC4	TIM3 OC4 selected as blanking source for comparator
COMP_BLANKINGSRCE_TIM15OC1	TIM15 OC1 selected as blanking source for comparator
COMP_BLANKINGSRCE_TIM15OC2	TIM15 OC2 selected as blanking source for comparator
IS_COMP_BLANKINGSRCE	
IS_COMP_BLANKINGSRCE_INSTANCE	
COMP_CSR_COMPxBLANKING_MASK	COMP_CSR_COMPxBLANKING mask

COMP Extended Exported Constants

COMP_CSR_RESET_VALUE	
COMP_CSR_COMPxINSEL_MASK	COMP_CSR_COMPxINSEL Mask
COMP_CSR_COMPxOUTSEL_MASK	COMP_CSR_COMPxOUTSEL Mask
COMP_CSR_COMPxPOL_MASK	COMP_CSR_COMPxPOL Mask

COMP Extended EXTI Line Event (STM32F303xE/STM32F398xx/STM32F303xC/STM32F358xx Product devices)

COMP_EXTI_LINE_MASK	Mask on possible line values
COMP_EXTI_LINE_REG_MASK	Mask on possible register values
COMP_EXTI_LINE_COMP1_EVENT	External interrupt line 21 Connected to COMP1

COMP_EXTI_LINE_COMP2_EVENT	External interrupt line 22 Connected to COMP2
COMP_EXTI_LINE_COMP3_EVENT	External interrupt line 29 Connected to COMP3
COMP_EXTI_LINE_COMP4_EVENT	External interrupt line 30 Connected to COMP4
COMP_EXTI_LINE_COMP5_EVENT	External interrupt line 31 Connected to COMP5
COMP_EXTI_LINE_COMP6_EVENT	External interrupt line 32 Connected to COMP6
COMP_EXTI_LINE_COMP7_EVENT	External interrupt line 33 Connected to COMP7

COMP Extended Hysteresis

COMP_HYSTERESIS_NONE	No hysteresis
IS_COMP_HYSTERESIS	Not available: check always true
COMP_CSR_COMPxHYST_MASK	Mask empty: feature not available

COMP Extended InvertingInput (STM32F302xE/STM32F303xE/STM32F398xx/STM32F302xC/STM32F303xC/STM32F358xx Product devices)

COMP_INVERTINGINPUT_1_4VREFINT	1/4 VREFINT connected to comparator inverting input
COMP_INVERTINGINPUT_1_2VREFINT	1/2 VREFINT connected to comparator inverting input
COMP_INVERTINGINPUT_3_4VREFINT	3/4 VREFINT connected to comparator inverting input
COMP_INVERTINGINPUT_VREFINT	VREFINT connected to comparator inverting input
COMP_INVERTINGINPUT_DAC1_CH1	DAC1_CH1_OUT (PA4) connected to comparator inverting input
COMP_INVERTINGINPUT_DAC1_CH2	DAC1_CH2_OUT (PA5) connected to comparator inverting input
COMP_INVERTINGINPUT_IO1	IO1 (PA0 for COMP1, PA2 for COMP2, PD15 for COMP3, PE8 for COMP4, PD13 for COMP5, PD10 for COMP6, PC0 for COMP7) connected to comparator inverting input
COMP_INVERTINGINPUT_IO2	IO2 (PB12 for COMP3, PB2 for COMP4, PB10 for COMP5, PB15 for COMP6) connected to comparator inverting input
COMP_INVERTINGINPUT_DAC1	
COMP_INVERTINGINPUT_DAC2	
IS_COMP_INVERTINGINPUT	

COMP Extended Mode

IS_COMP_MODE	< Power mode not available Not available: check always true
COMP_CSR_COMPxMODE_MASK	Mask empty: feature not available

COMP Extended NonInvertingInput (STM32F302xE/STM32F303xE/STM32F398xx Product devices)

COMP_NONINVERTINGINPUT_IO1	IO1 (PA1 for COMP1, PA7 for COMP2, PB14 for COMP3, PB0 for COMP4, PD12 for COMP5, PD11 for COMP6, PA0 for COMP7) connected to comparator non inverting input
COMP_NONINVERTINGINPUT_DAC1SWITCHCLOSED	DAC ouput connected to comparator COMP1 non inverting input
IS_COMP_NONINVERTINGINPUT	
IS_COMP_NONINVERTINGINPUT_INSTANCE	
COMP_CSR_COMPxNONINSEL_MASK	COMP_CSR_COMPxNONINSEL mask

COMP Extended Output (STM32F303xE/STM32F398xx Product devices)

COMP_OUTPUT_NONE	COMP output isn't connected to other peripherals
COMP_OUTPUT_TIM1BKIN	COMP output connected to TIM1 Break Input (BKIN)
COMP_OUTPUT_TIM1BKIN2	COMP output connected to TIM1 Break Input 2 (BKIN2)
COMP_OUTPUT_TIM8BKIN	COMP output connected to TIM8 Break Input (BKIN)
COMP_OUTPUT_TIM8BKIN2	COMP output connected to TIM8 Break Input 2 (BKIN2)
COMP_OUTPUT_TIM1BKIN2_TIM8BKIN2	COMP output connected to TIM1 Break Input 2 and TIM8 Break Input 2
COMP_OUTPUT_TIM20BKIN	COMP output connected to TIM20 Break Input (BKIN)
COMP_OUTPUT_TIM20BKIN2	COMP output connected to TIM20 Break Input 2 (BKIN2)
COMP_OUTPUT_TIM1BKIN2_TIM8BKIN2_TIM20BKIN2	COMP output connected to TIM1 Break Input 2, TIM8 Break Input 2 and TIM20 Break Input 2
COMP_OUTPUT_TIM1OCREFCLR	COMP output connected to TIM1 OCREF Clear
COMP_OUTPUT_TIM1IC1	COMP output connected to TIM1 Input Capture 1
COMP_OUTPUT_TIM2IC4	COMP output connected to TIM2 Input Capture 4
COMP_OUTPUT_TIM2OCREFCLR	COMP output connected to TIM2 OCREF Clear
COMP_OUTPUT_TIM3IC1	COMP output connected to TIM3 Input Capture 1

COMP_OUTPUT_TIM3OCREFCLR	COMP output connected to TIM3 OCREF Clear
COMP_OUTPUT_TIM20OCREFCLR	COMP output connected to TIM20 OCREF Clear
COMP_OUTPUT_TIM4IC1	COMP output connected to TIM4 Input Capture 1
COMP_OUTPUT_TIM3IC2	COMP output connected to TIM3 Input Capture 2
COMP_OUTPUT_TIM15IC1	COMP output connected to TIM15 Input Capture 1
COMP_OUTPUT_TIM15BKIN	COMP output connected to TIM15 Break Input (BKIN)
COMP_OUTPUT_TIM3IC3	COMP output connected to TIM3 Input Capture 3
COMP_OUTPUT_TIM8OCREFCLR	COMP output connected to TIM8 OCREF Clear
COMP_OUTPUT_TIM15IC2	COMP output connected to TIM15 Input Capture 2
COMP_OUTPUT_TIM4IC2	COMP output connected to TIM4 Input Capture 2
COMP_OUTPUT_TIM15OCREFCLR	COMP output connected to TIM15 OCREF Clear
COMP_OUTPUT_TIM2IC1	COMP output connected to TIM2 Input Capture 1
COMP_OUTPUT_TIM17IC1	COMP output connected to TIM17 Input Capture 1
COMP_OUTPUT_TIM4IC3	COMP output connected to TIM4 Input Capture 3
COMP_OUTPUT_TIM16BKIN	COMP output connected to TIM16 Break Input (BKIN)
COMP_OUTPUT_TIM2IC2	COMP output connected to TIM2 Input Capture 2
COMP_OUTPUT_COMP6TIM2OCREFCLR	COMP output connected to TIM2 OCREF Clear
COMP_OUTPUT_TIM16OCREFCLR	COMP output connected to TIM16 OCREF Clear
COMP_OUTPUT_TIM16IC1	COMP output connected to TIM16 Input Capture 1
COMP_OUTPUT_TIM4IC4	COMP output connected to TIM4 Input Capture 4
COMP_OUTPUT_TIM2IC3	COMP output connected to TIM2 Input Capture 3
COMP_OUTPUT_TIM1IC2	COMP output connected to TIM1 Input Capture 2

COMP_OUTPUT_TIM17OCREFCLR	COMP output connected to TIM16 OCREF Clear
COMP_OUTPUT_TIM17BKIN	COMP output connected to TIM16 Break Input (BKIN)
IS_COMP_OUTPUT	

COMP Extended WindowMode (STM32F302xE/STM32F303xE/STM32F398xx Product devices)

COMP_WINDOWMODE_DISABLED	Window mode disabled
COMP_WINDOWMODE_ENABLED	Window mode enabled: non inverting input of comparator X (x=2,4,6) is connected to the non inverting input of comparator X-1
IS_COMP_WINDOWMODE	
COMP_CSR_COMPxWNDWEN_MASK	COMP_CSR_COMPxWNDWEN mask

10 HAL CORTEX Generic Driver

10.1 CORTEX Firmware driver API description

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

10.1.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

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

1. Configure the NVIC Priority Grouping using HAL_NVIC_SetPriorityGrouping() function according to *Table 21: "Pre-emption priority and subpriority vs Priority Grouping configuration"*. @brief CORTEX_NVIC_Priority_Table. gives the allowed values of the pre-emption priority and subpriority according to the Priority Grouping configuration performed by HAL_NVIC_SetPriorityGrouping() function.
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority()
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ() When the NVIC_PRIORITYGROUP_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the sub priority. IRQ priority order (sorted by highest to lowest priority): Lowest pre-emption priority Lowest sub priority Lowest hardware priority (IRQ number)

Table 21: Pre-emption priority and subpriority vs Priority Grouping configuration

NVIC_PriorityGroup	NVIC_IRQChannelPreemptionPriority	NVIC_IRQChannelSubPriority	Description
NVIC_PRIORITYGROUP_0	0	0-15	0 bit for pre-emption priority 4 bits for subpriority
NVIC_PRIORITYGROUP_1	0-1	0-7	1 bit for pre-emption priority 3 bits for subpriority
NVIC_PRIORITYGROUP_2	0-3	0-3	2 bits for pre-emption priority 2 bits for subpriority
NVIC_PRIORITYGROUP_3	0-7	0-1	3 bits for pre-emption priority 1 bit for subpriority

NVIC_PriorityGroup	NVIC_IRQChannelPreemptionPriority	NVIC_IRQChannelSubPriority	Description
NVIC_PRIORITYGROUP_4	0-15	0	4 bits for pre-emption priority 0 bit for subpriority

How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for time base

- The HAL_SYSTICK_Config()function calls the SysTick_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x0F).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro __HAL_CORTEX_SYSTICKCLK_CONFIG(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The __HAL_CORTEX_SYSTICKCLK_CONFIG() macro is defined inside the stm32f3xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

10.1.2 Initialization and de-initialization functions

This section provides 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()***

10.1.3 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()*

10.1.4 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.

10.1.5 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 (stm32f3xxxx.h)) • **PreemptPriority:** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 as described in the table CORTEX_NVIC_Priority_Table 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 as described in the table CORTEX_NVIC_Priority_Table A lower priority value indicates a higher priority.
Return values	• None

10.1.6 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 (stm32f3xxxx.h))
Return values	• None
Notes	• To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

10.1.7 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 (stm32f3xxxx.h))
Return values	• None

10.1.8 HAL_NVIC_SystemReset

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

10.1.9 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.

10.1.10 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)

10.1.11 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 (stm32f3xxxx.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

10.1.12 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 (stm32f3xxxx.h))
Return values	• None

10.1.13 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 (stm32f3xxxx.h))
Return values	•	status - 0 Interrupt status is not pending. 1 Interrupt status is pending.

10.1.14 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 (stm32f3xxxx.h))
Return values	• None

10.1.15 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 (stm32f3xxxx.h))
Return values	• status - 0 Interrupt status is not pending. 1 Interrupt status is pending.

10.1.16 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

10.1.17 HAL_SYSTICK_IRQHandler

Function Name	**void HAL_SYSTICK_IRQHandler (void)**

Function Description	This function handles SYSTICK interrupt request.
Return values	• None

10.1.18 HAL_SYSTICK_Callback

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

10.2 CORTEX Firmware driver defines

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

10.2.1 CORTEX

CORTEX

CORTEX Exported Macros

__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 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
IS_NVIC_PRIORITY_GROUP	
IS_NVIC_PREEMPTION_PRIORITY	
IS_NVIC_SUB_PRIORITY	

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

IS_SYSTICK_CLK_SOURCE

11 HAL CRC Generic Driver

11.1 CRC Firmware driver registers structures

11.1.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the stm32f3xx_hal_crc.h

Data Fields

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

Field Documentation

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

0x1A2B3C4D becomes 0xD458B23C CRC_INPUTDATA_INVERSION_WORD word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

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

11.1.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32f3xx_hal_crc.h

Data Fields

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

Field Documentation

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

11.2 CRC Firmware driver API description

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

11.2.1 How to use this driver

1. Enable CRC AHB clock using __CRC_CLK_ENABLE();
2. Initialize CRC calculator

  - specify generating polynomial (IP default or non-default one)
  - specify initialization value (IP default or non-default one)
  - specify input data format
  - specify input or output data inversion mode if any
3. Use HAL_CRC_Accumulate() function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value
4. Use HAL_CRC_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

11.2.2 Initialization and Configuration 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()***

11.2.3 Peripheral Control functions

This section provides functions allowing to:

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

11.2.4 Peripheral State functions

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

- ***HAL_CRC_GetState()***

11.2.5 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:** CRC handle
Return values	• HAL status

11.2.6 HAL_CRC_DeInit

Function Name	**HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)**
Function Description	DeInitializes the CRC peripheral.
Parameters	• **hcrc:** CRC handle
Return values	• HAL status

11.2.7 HAL_CRC_MspInit

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

11.2.8 HAL_CRC_MspDeInit

Function Name	**void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)**
Function Description	DeInitializes the CRC MSP.
Parameters	• **hcrc:** CRC handle
Return values	• None

11.2.9 HAL_CRC_Accumulate

Function Name	**uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)**
Function Description	Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.
Parameters	• **hcrc:** CRC handle • **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat. • **BufferLength:** input data buffer length
Return values	• uint32_t CRC (returned value LSBs for CRC shorter than 32 bits)

11.2.10 HAL_CRC_Calculate

Function Name	**uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)**

Function Description	Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.
Parameters	• **hcrc:** CRC handle • **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat. • **BufferLength:** input data buffer length
Return values	• uint32_t CRC (returned value LSBs for CRC shorter than 32 bits)

11.2.11 HAL_CRC_GetState

Function Name	**HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)**
Function Description	Returns the CRC state.
Parameters	• **hcrc:** CRC handle
Return values	• HAL state

11.3 CRC Firmware driver defines

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

11.3.1 CRC

CRC

Default CRC computation initialization value

DEFAULT_CRC_INITVALUE

Indicates whether or not default init value is used

DEFAULT_INIT_VALUE_ENABLE

DEFAULT_INIT_VALUE_DISABLE

IS_DEFAULT_INIT_VALUE

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

DEFAULT_POLYNOMIAL_DISABLE

IS_DEFAULT_POLYNOMIAL

Default CRC generating polynomial

DEFAULT_CRC32_POLY

CRC Exported Macros

__HAL_CRC_RESET_HANDLE_STATE	**Description:** • Reset CRC handle state. **Parameters:** • __HANDLE__: CRC handle.

Return value:

- None:

__HAL_CRC_DR_RESET

Description:

- Reset CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None.:

__HAL_CRC_INITIALCRCVALUE_CONFIG

Description:

- Set CRC INIT non-default value.

Parameters:

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

Return value:

- None.:

Input Buffer Format

CRC_INPUTDATA_FORMAT_UNDEFINED

CRC_INPUTDATA_FORMAT_BYTES

CRC_INPUTDATA_FORMAT_HALFWORDS

CRC_INPUTDATA_FORMAT_WORDS

IS_CRC_INPUTDATA_FORMAT

Polynomial sizes to configure the IP

CRC_POLYLENGTH_32B

CRC_POLYLENGTH_16B

CRC_POLYLENGTH_8B

CRC_POLYLENGTH_7B

IS_CRC_POL_LENGTH

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

HAL_CRC_LENGTH_16B

HAL_CRC_LENGTH_8B

HAL_CRC_LENGTH_7B

12 HAL CRC Extension Driver

12.1 CRCEx Firmware driver API description

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

12.1.1 Product specific features

12.1.2 How to use this driver

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

12.1.3 HAL_CRCEx_Polynomial_Set

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

12.1.4 HAL_CRCEx_Input_Data_Reverse

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

12.1.5 HAL_CRCEx_Output_Data_Reverse

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

12.2 CRCEx Firmware driver defines

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

12.2.1 CRCEx

CRCEx

CRC Extended Exported Macros

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

Parameters:

- __HANDLE__: CRC handle

Return value:

- None.:

__HAL_CRC_POLYNOMIAL_CONFIG

Description:

- Set CRC non-default polynomial.

Parameters:

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

Return value:

- None.:

CRC Extended Input Data Inversion Modes

CRC_INPUTDATA_INVERSION_NONE

CRC_INPUTDATA_INVERSION_BYTE

CRC_INPUTDATA_INVERSION_HALFWORD

CRC_INPUTDATA_INVERSION_WORD

IS_CRC_INPUTDATA_INVERSION_MODE

CRC Extended Output Data Inversion Modes

CRC_OUTPUTDATA_INVERSION_DISABLED

CRC_OUTPUTDATA_INVERSION_ENABLED

IS_CRC_OUTPUTDATA_INVERSION_MODE

13 HAL DAC Generic Driver

13.1 DAC Firmware driver registers structures

13.1.1 DAC_ChannelConfTypeDef

DAC_ChannelConfTypeDef is defined in the stm32f3xx_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***
- ***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***

13.1.2 __DAC_HandleTypeDef

__DAC_HandleTypeDef is defined in the stm32f3xx_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

13.2 DAC Firmware driver API description

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

13.2.1 DAC Peripheral features

DAC Channels

The device integrates up to 3 12-bit Digital Analog Converters that can be used independently or simultaneously (dual mode):

1. DAC1 channel1 with DAC1_OUT1 (PA4) as output
2. DAC1 channel2 with DAC1_OUT2 (PA5) as output (for STM32F3 devices having 2 channels on DAC1)
3. DAC2 channel1 with DAC2_OUT1 (PA6) as output (for STM32F3 devices having 2 DAC)

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_Trigger_None and DAC1_OUT1/DAC1_OUT2/DAC2_OUT1 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_Pin9) using DAC_Trigger_Ext_IT9. The used pin (GPIOx_Pin9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM4, TIM5, TIM6, TIM7 and TIM8 (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 wave generation feature

Both DAC channels of DAC1 can be used to generate note that wave generation is not available in DAC2.

1. Noise wave
2. Triangle wave Wave generation is NOT available in DAC2.

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation: DAC_OUTx = VREF+ * DOR / 4095 with DOR is the Data Output Register VEF+ is the input voltage reference (refer to the device datasheet) e.g. To set DAC_OUT1 to 0.7V, use Assuming that VREF+ = 3.3V, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

A DMA1 or DMA2 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 or DMA2 requests are enabled using HAL_DAC_Start_DMA()

DMA1 requests are mapped as following:

1. DAC1 channel1: mapped either on - DMA1 channel3 - or DMA2 channel3 (for STM32F3 devices having 2 DMA) which must be already configured
2. DAC1 channel2: (for STM32F3 devices having 2 channels on DAC1) mapped either on - DMA1 channel4 - or DMA2 channel4 (for STM32F3 devices having 2 DMA) which must be already configured
3. DAC2 channel1: mapped either on (for STM32F3 devices having 2 DAC) - DMA1 channel4 - or DMA2 channel4 (for STM32F3 devices having 2 DMA) which must be already configured

13.2.2 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 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 transfered at each end of conversion
- 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() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

DAC HAL driver macros list

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

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __HAL_DAC_DISABLE : Disable the DAC peripheral
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status

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

13.2.3 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()***

13.2.4 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.
- Get result of dual mode conversion.
- ***HAL_DAC_Start()***
- ***HAL_DAC_Stop()***
- ***HAL_DAC_Stop_DMA()***
- ***HAL_DAC_GetValue()***
- ***HAL_DACEx_DualGetValue()***
- ***HAL_DAC_ConvCpltCallbackCh1()***
- ***HAL_DAC_ConvHalfCpltCallbackCh1()***
- ***HAL_DAC_ErrorCallbackCh1()***
- ***HAL_DAC_DMAUnderrunCallbackCh1()***
- ***HAL_DAC_Start_DMA()***
- ***HAL_DAC_IRQHandler()***

13.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Configure Triangle wave generation.
- Configure Noise wave generation.
- Set the specified data holding register value for DAC channel.
- Set the specified data holding register value for Dual DAC channels.
- *HAL_DAC_ConfigChannel()*
- *HAL_DAC_SetValue()*
- *HAL_DACEx_DualSetValue()*

13.2.6 DAC Peripheral State and Error functions

This subsection provides functions allowing to

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

13.2.7 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

13.2.8 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

13.2.9 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

13.2.10 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

13.2.11 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: DAC1_CHANNEL_1: DAC1 Channel1 selected DAC1_CHANNEL_2: DAC1 Channel2 selected DAC2_CHANNEL_1: DAC2 Channel1 selected
Return values	• HAL status

13.2.12 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: DAC1_CHANNEL_1: DAC1 Channel1 selected DAC1_CHANNEL_2: DAC1 Channel2 selected DAC2_CHANNEL_1: DAC2 Channel1 selected
Return values	• HAL status

13.2.13 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: DAC1_CHANNEL_1: DAC1 Channel1 selected DAC1_CHANNEL_2: DAC1 Channel2 selected DAC2_CHANNEL_1: DAC2 Channel1 selected
Return values	• HAL status

13.2.14 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: DAC1_CHANNEL_1: DAC1 Channel1 selected DAC1_CHANNEL_2: DAC1 Channel2 selected DAC2_CHANNEL_1: DAC2 Channel1 selected
Return values	• The selected DAC channel data output value.

13.2.15 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.

13.2.16 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

13.2.17 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

13.2.18 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

13.2.19 HAL_DAC_DMAUnderrunCallbackCh1

Function Name	**void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)**
Function Description	DMA underrun DAC callback for Channel1.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None

13.2.20 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: DAC1 Channel1 selected DAC_CHANNEL_2: DAC1 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

13.2.21 HAL_DAC_IRQHandler

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

13.2.22 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: DAC1_CHANNEL_1: DAC1 Channel1 selected DAC1_CHANNEL_2: DAC1 Channel2 selected DAC2_CHANNEL_1: DAC2 Channel1 selected
Return values	• HAL status

13.2.23 HAL_DAC_SetValue

Function Name	**HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t channel, uint32_t alignment, uint32_t data)**
Function Description	

13.2.24 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 • **data2:** Data for DAC Channel2 to be loaded in the selected data holding register. • **data1:** 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.

13.2.25 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

13.2.26 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

13.3 DAC Firmware driver defines

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

13.3.1 DAC

DAC

DAC data

IS_DAC_DATA

DAC data alignement

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

IS_DAC_ALIGN

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__: DAC handle.

Return value:

- None:

__HAL_DAC_ENABLE

__HAL_DAC_DISABLE

__HAL_DHR12R1_ALIGNEMENT

__HAL_DHR12R2_ALIGNEMENT

__HAL_DHR12RD_ALIGNEMENT

__HAL_DAC_ENABLE_IT

__HAL_DAC_DISABLE_IT

__HAL_DAC_GET_FLAG

__HAL_DAC_CLEAR_FLAG

DAC flags definition

DAC_FLAG_DMAUDR1

DAC_FLAG_DMAUDR2

IS_DAC_FLAG

DAC interrupts definition

DAC_IT_DMAUDR1

DAC_IT_DMAUDR2

IS_DAC_IT

DAC 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

IS_DAC_LFSR_UNMASK_TRIANGLE_AMPLITUDE

DAC output buffer

DAC_OUTPUTBUFFER_ENABLE

DAC_OUTPUTBUFFER_DISABLE

IS_DAC_OUTPUT_BUFFER_STATE

DAC wave generation

DAC_WAVEGENERATION_NONE

DAC_WAVEGENERATION_NOISE

DAC_WAVEGENERATION_TRIANGLE

IS_DAC_GENERATE_WAVE

DAC_WAVE_NOISE

DAC_WAVE_TRIANGLE

IS_DAC_WAVE

14 HAL DAC Extension Driver

14.1 DACEx Firmware driver API description

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

14.1.1 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.

14.1.2 Peripheral Control functions

This section provides functions allowing to:

- Set the specified data holding register value for DAC channel.
- Set the specified data holding register value for dual DAC channel (when DAC channel 2 is present in DAC 1)
- ***HAL_DAC_SetValue()***
- ***HAL_DACEx_DualSetValue()***

14.1.3 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Start conversion and enable DMA transfer.
- Get result of conversion.
- Handle DAC IRQ's.
- Generate triangular-wave
- Generate noise-wave
- Callback functions for DAC1 Channel2 (when supported)
- ***HAL_DAC_Start()***
- ***HAL_DAC_Start_DMA()***
- ***HAL_DAC_GetValue()***
- ***HAL_DACEx_DualGetValue()***
- ***HAL_DAC_IRQHandler()***
- ***HAL_DACEx_TriangleWaveGenerate()***
- ***HAL_DACEx_NoiseWaveGenerate()***
- ***HAL_DACEx_ConvCpltCallbackCh2()***
- ***HAL_DACEx_ConvHalfCpltCallbackCh2()***
- ***HAL_DACEx_ErrorCallbackCh2()***
- ***HAL_DACEx_DMAUnderrunCallbackCh2()***

14.1.4 HAL_DAC_SetValue

Function Name	**HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t channel, uint32_t alignment, uint32_t data)**
Function Description	

14.1.5 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 • **data2:** Data for DAC Channel2 to be loaded in the selected data holding register. • **data1:** 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.

14.1.6 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: DAC1_CHANNEL_1: DAC1 Channel1 selected DAC1_CHANNEL_2: DAC1 Channel2 selected DAC2_CHANNEL_1: DAC2 Channel1 selected
Return values	• HAL status

14.1.7 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: DAC1 Channel1 selected DAC_CHANNEL_2: DAC1 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

14.1.8 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: DAC1_CHANNEL_1: DAC1 Channel1 selected DAC1_CHANNEL_2: DAC1 Channel2 selected DAC2_CHANNEL_1: DAC2 Channel1 selected
Return values	• The selected DAC channel data output value.

14.1.9 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.

14.1.10 HAL_DAC_IRQHandler

Function Name	**void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)**
Function Description	Handles DAC interrupt request.

Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None

14.1.11 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: DAC1 Channel1 selected DAC_CHANNEL_2: DAC1 Channel2 selected • **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values: DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1 DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3 DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7 DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15 DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31 DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63 DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127 DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255 DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511 DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023 DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047 DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095
Return values	• HAL status
Notes	• Wave generation is not available in DAC2.

14.1.12 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: DAC1 Channel1 selected DAC_CHANNEL_2: DAC1 Channel2 selected • **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values: DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation
Return values	• HAL status

14.1.13 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

14.1.14 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

14.1.15 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

14.1.16 HAL_DACEx_DMAUnderrunCallbackCh2

Function Name	**void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)**
Function Description	DMA underrun DAC callback for channel2.
Parameters	• **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	• None

14.2 DACEx Firmware driver defines

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

14.2.1 DACEx

DACEx

DAC Extended Channel selection

DAC_CHANNEL_1	DAC Channel 1
DAC_CHANNEL_2	DAC Channel 2
DAC1_CHANNEL_1	DAC1 Channel 1
DAC1_CHANNEL_2	DAC1 Channel 2
IS_DAC_CHANNEL	

DAC Extended trigger selection

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
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_T15_TRGO	TIM5 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T3_TRGO	TIM3 TRGO selected as external conversion trigger for DAC channel Use
DAC_TRIGGER_T8_TRGO	TIM8 TRGO selected as external conversion trigger for DAC channel Use
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
IS_DAC_TRIGGER	

15 HAL DMA Generic Driver

15.1 DMA Firmware driver registers structures

15.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32f3xx_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***

15.1.2 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32f3xx_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

15.2 DMA Firmware driver API description

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

15.2.1 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

15.2.2 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()*

15.2.3 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()*

15.2.4 State and Errors functions

This subsection provides functions allowing to

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

15.2.5 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

15.2.6 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

15.2.7 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

15.2.8 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

15.2.9 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.

15.2.10 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

15.2.11 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

15.2.12 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

15.2.13 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

15.3 DMA Firmware driver defines

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

15.3.1 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 Code

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

DMA Remap Enable

__HAL_REMAPDMA_CHANNEL_ENABLE	**Description:** • DMA remapping enable/disable macros. **Parameters:** • __DMA_REMAP__: This parameter can be a value of
__HAL_REMAPDMA_CHANNEL_DISABLE	

16 HAL DMA Extension Driver

16.1 DMAEx Firmware driver defines

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

16.1.1 DMAEx

DMAEx

DMA Extended Exported Macros

__HAL_DMA_GET_TC_FLAG_INDEX

Description:

- Returns the current DMA Channel transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer complete flag index.

__HAL_DMA_GET_HT_FLAG_INDEX

Description:

- 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:

17 HAL FLASH Generic Driver

17.1 FLASH Firmware driver registers structures

17.1.1 FLASH_EraseInitTypeDef

FLASH_EraseInitTypeDef is defined in the stm32f3xx_hal_flash.h

Data Fields

- ***uint32_t TypeErase***
- ***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 ***FLASH_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::PageAddress***
  PageAdress: Initial FLASH page address to erase when mass erase is disabled This parameter must be a value of ***FLASHEx_Address***
- ***uint32_t FLASH_EraseInitTypeDef::NbPages***
  NbPages: Number of pagess to be erased. This parameter must be a value between 1 and (max number of pages - value of initial page)

17.1.2 FLASH_OBProgramInitTypeDef

FLASH_OBProgramInitTypeDef is defined in the stm32f3xx_hal_flash.h

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPState***
- ***uint32_t WRPPage***
- ***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 ***FLASH_OB_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPState***
  WRPState: Write protection activation or deactivation. This parameter can be a value of ***FLASH_OB_WRP_State***

- ***uint32_t FLASH_OBProgramInitTypeDef::WRPPage***
  WRPSector: specifies the page(s) to be write protected This parameter can be a value of ***FLASHEx_OB_Write_Protection***
- ***uint8_t FLASH_OBProgramInitTypeDef::RDPLevel***
  RDPLevel: Set the read protection level.. This parameter can be a value of ***FLASH_OB_Read_Protection***
- ***uint8_t FLASH_OBProgramInitTypeDef::USERConfig***
  USERConfig: Program the FLASH User Option Byte: IWDG / STOP / STDBY / BOOT1 / VDDA_ANALOG / SRAM_PARITY / SDADC12_VDD_MONITOR This parameter can be a combination of ***FLASH_OB_IWatchdog***, ***FLASH_OB_nRST_STOP***, ***FLASH_OB_nRST_STDBY***, ***FLASH_OB_BOOT1***, ***FLASH_OB_VDDA_Analog_Monitoring***, ***FLASH_OB_SRAM_Parity_Enable*** and ***FLASH_OB_SDADC12_VDD_MONITOR***
- ***uint32_t FLASH_OBProgramInitTypeDef::DATAAddress***
  DATAAddress: Address of the option byte DATA to be prgrammed This parameter can be a value of ***FLASH_OB_Data_Address***
- ***uint8_t FLASH_OBProgramInitTypeDef::DATAData***
  DATAData: Data to be stored in the option byte DATA This parameter can have any value

17.1.3 FLASH_ProcessTypeDef

FLASH_ProcessTypeDef is defined in the stm32f3xx_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 FLASH_ErrorTypeDef 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 FLASH_ErrorTypeDef FLASH_ProcessTypeDef::ErrorCode***

17.2 FLASH Firmware driver API description

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

17.2.1 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:

1. Flash memory read operations
2. Flash memory program/erase operations
3. Read / write protections
4. Prefetch on I-Code

17.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32F3xx 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 and word
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

17.2.3 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()*

17.2.4 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()*

17.2.5 Peripheral State functions

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

- *HAL_FLASH_GetError()*

17.2.6 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 FLASH 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.

17.2.7 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 FLASH 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.

17.2.8 HAL_FLASH_IRQHandler

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

17.2.9 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

17.2.10 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

17.2.11 HAL_FLASH_Unlock

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

17.2.12 HAL_FLASH_Lock

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

17.2.13 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

17.2.14 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

17.2.15 HAL_FLASH_OB_Launch

Function Name	**HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)**
Function Description	Launch the option byte loading.
Return values	• HAL status

17.2.16 HAL_FLASH_GetError

Function Name	**FLASH_ErrorTypeDef 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

17.3 FLASH Firmware driver defines

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

17.3.1 FLASH

FLASH

FLASH Flag definition

FLASH_FLAG_BSY	FLASH Busy flag
FLASH_FLAG_PGERR	FLASH Programming error flag
FLASH_FLAG_WRPERR	FLASH Write protected error flag
FLASH_FLAG_EOP	FLASH End of Operation flag
IS_FLASH_CLEAR_FLAG	
IS_FLASH_GET_FLAG	

FLASH Half Cycle

__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:

FLASH 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: End of FLASH Operation Interrupt
  - FLASH_IT_ERR: Error Interrupt

Return value:

- none:

__HAL_FLASH_DISABLE_IT

Description:

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:

- FLASH_IT_EOP: End of FLASH Operation Interrupt
- FLASH_IT_ERR: Error Interrupt

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 : FLASH End of Operation flag
  - FLASH_FLAG_WRPERR: FLASH Write protected error flag
  - FLASH_FLAG_PGERR : FLASH Programming error flag
  - FLASH_FLAG_BSY : FLASH Busy flag

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 : FLASH End of Operation flag
  - FLASH_FLAG_WRPERR: FLASH Write protected error flag
  - FLASH_FLAG_PGERR : FLASH Programming error flag

Return value:

- none:

FLASH Interrupt definition

FLASH_IT_EOP	End of FLASH Operation Interrupt source
FLASH_IT_ERR	Error Interrupt source
IS_FLASH_IT	

FLASH Latency

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

__HAL_FLASH_SET_LATENCY	**Description:** • Set the FLASH Latency. **Parameters:** • __LATENCY__: FLASH Latency The value of this parameter depend on device used within the same series **Return value:** • None:

FLASH Option Byte BOOT1

OB_BOOT1_RESET	BOOT1 Reset
OB_BOOT1_SET	BOOT1 Set
IS_OB_BOOT1	

FLASH Option Byte Data Address

IS_OB_DATA_ADDRESS

FLASH Option Byte IWatchdog

OB_IWDG_SW	Software IWDG selected
OB_IWDG_HW	Hardware IWDG selected
IS_OB_IWDG_SOURCE	

FLASH Option Byte nRST STDBY

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

FLASH Option Byte nRST STOP

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

FLASH Option Byte Read Protection

OB_RDP_LEVEL_0	
OB_RDP_LEVEL_1	
OB_RDP_LEVEL_2	Warning: When enabling read protection level 2 it's no more possible to go back to level 1 or 0
IS_OB_RDP_LEVEL	

FLASH Option Byte SDADC12 VDD MONITOR

OB_SDADC12_VDD_MONITOR_SET	SDADC12_VDD power supply supervisor set
OB_SDADC12_VDD_MONITOR_RESET	SDADC12_VDD power supply supervisor reset
IS_OB_SDADC12_VDD_MONITOR	

FLASH Option Byte SRAM Parity Enable

OB_SRAM_PARITY_SET	SRAM parity enable set
OB_SRAM_PARITY_RESET	SRAM parity enable reset
IS_OB_SRAM_PARITY	

FLASH 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
IS_OPTIONBYTE	

FLASH Option Byte VDDA Analog Monitoring

OB_VDDA_ANALOG_ON	Analog monitoring on VDDA Power source ON
OB_VDDA_ANALOG_OFF	Analog monitoring on VDDA Power source OFF
IS_OB_VDDA_ANALOG	

FLASH WRP State

WRPSTATE_DISABLE	Disable the write protection of the desired pages
WRPSTATE_ENABLE	Enable the write protection of the desired pagess
IS_WRPSTATE	

FLASH Prefetch

__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 Define

HAL_FLASH_TIMEOUT_VALUE

FLASH Timeout definition

HAL_FLASH_TIMEOUT_VALUE

FLASH Type Erase

TYPEERASE_PAGES	Pages erase only
TYPEERASE_MASSERASE	Flash mass erase activation
IS_TYPEERASE	

FLASH Type Program

TYPEPROGRAM_HALFWORD	Program a half-word (16-bit) at a specified address.
TYPEPROGRAM_WORD	Program a word (32-bit) at a specified address.
TYPEPROGRAM_DOUBLEWORD	Program a double word (64-bit) at a specified address
IS_TYPEPROGRAM	

18 HAL FLASH Extension Driver

18.1 FLASHEx Firmware driver API description

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

18.1.1 IO operation functions

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

18.1.2 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()*

18.1.3 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

18.1.4 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

18.1.5 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)

18.1.6 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)

18.1.7 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

18.2 FLASHEx Firmware driver defines

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

18.2.1 FLASHEx

FLASHEx

FLASH Extended Address

IS_FLASH_PROGRAM_ADDRESS

FLASH Extended Exported Constants

FLASH_SIZE_DATA_REGISTER

FLASH_PAGE_SIZE

FLASH Extended Nb Pages

IS_FLASH_NB_PAGES

FLASH Extended Option Bytes Write Protection

OB_WRP_PAGES0TO1

OB_WRP_PAGES2TO3

OB_WRP_PAGES4TO5

OB_WRP_PAGES6TO7

OB_WRP_PAGES8TO9

OB_WRP_PAGES10TO11

OB_WRP_PAGES12TO13

OB_WRP_PAGES14TO15

OB_WRP_PAGES16TO17

OB_WRP_PAGES18TO19

OB_WRP_PAGES20TO21

OB_WRP_PAGES22TO23

OB_WRP_PAGES24TO25

OB_WRP_PAGES26TO27

OB_WRP_PAGES28TO29

OB_WRP_PAGES30TO31

OB_WRP_PAGES32TO33

OB_WRP_PAGES34TO35

OB_WRP_PAGES36TO37

OB_WRP_PAGES38TO39

OB_WRP_PAGES40TO41

OB_WRP_PAGES42TO43

OB_WRP_PAGES44TO45

OB_WRP_PAGES46TO47

OB_WRP_PAGES48TO49

OB_WRP_PAGES50TO51

OB_WRP_PAGES52TO53

OB_WRP_PAGES54TO55

OB_WRP_PAGES56TO57

OB_WRP_PAGES58TO59

OB_WRP_PAGES60TO61

OB_WRP_PAGES62TO255

OB_WRP_PAGES0TO15MASK

OB_WRP_PAGES16TO31MASK

OB_WRP_PAGES32TO47MASK

OB_WRP_PAGES48TO255MASK

OB_WRP_ALLPAGES	Write protection of all pages

IS_OB_WRP

FLASH Extended Private Define

HAL_FLASH_TIMEOUT_VALUE

19 HAL GPIO Generic Driver

19.1 GPIO Firmware driver registers structures

19.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32f3xx_hal_gpio.h

Data Fields

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

Field Documentation

- ***uint32_t GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_pins_define***
- ***uint32_t GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_mode_define***
- ***uint32_t GPIO_InitTypeDef::Pull***
  Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of ***GPIO_pull_define***
- ***uint32_t GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_speed_define***
- ***uint32_t GPIO_InitTypeDef::Alternate***
  Peripheral to be connected to the selected pins This parameter can be a value of ***GPIOEx_Alternate_function_selection***

19.2 GPIO Firmware driver API description

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

19.2.1 GPIO Peripheral features

Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

- Input mode
- Analog mode
- Output mode
- Alternate function mode
- External interrupt/event lines

During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.

All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.

In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.

The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral s alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.

All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

The external interrupt/event controller consists of up to 23 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

19.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function : __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, the speed is configurable: Low, Medium and High.
   - If alternate mode is selected, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure
   - Analog mode is required when a pin is to be used as ADC channel or DAC output.
   - In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. HAL_GPIO_DeInit allows to set register values to their reset value. It's also recommended to use it to unconfigure pin which was used as an external interrupt or in event mode. That's the only way to reset corresponding bit in EXTI & SYSCFG registers.
5. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
6. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
7. To lock pin configuration until next reset use HAL_GPIO_LockPin().
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).

9. 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.
10. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose Px0 and Px1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

19.2.3 Initialization and de-initialization functions

- *HAL_GPIO_Init()*
- *HAL_GPIO_DeInit()*

19.2.4 IO operation functions

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

19.2.5 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..F) to select the GPIO peripheral for STM32F3 family devices • **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• None

19.2.6 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..F) to select the GPIO peripheral for STM32F30X device or STM32F37X device • **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

19.2.7 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..F) to select the GPIO peripheral for STM32F3 family • **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
Return values	• The input port pin value.

19.2.8 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..F) to select the GPIO peripheral for STM32F3 family • **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). • **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_PIN_RESET: to clear the port pin GPIO_PIN_SET: to set the port pin
Return values	• None
Notes	• This function uses GPIOx_BSRR 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.

19.2.9 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..F) to select the GPIO peripheral for STM32F3 family • **GPIO_Pin:** specifies the pins to be toggled.
Return values	• None

19.2.10 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..F) to select the GPIO peripheral for STM32F3 family • **GPIO_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
Return values	• None
Notes	• The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH. • The configuration of the locked GPIO pins can no longer be modified until the next reset.

19.2.11 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

19.2.12 HAL_GPIO_EXTI_Callback

Function Name	**void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)**
Function Description	EXTI line detection callbacks.
Parameters	• **GPIO_Pin:** Specifies the pins connected EXTI line
Return values	• None

19.3 GPIO Firmware driver defines

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

19.3.1 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_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
IS_GPIO_MODE	

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

IS_GPIO_PIN

GPIO pin actions

IS_GPIO_PIN_ACTION

GPIO Private Defines

GET_GPIO_SOURCE

GPIO_MODE

EXTI_MODE

GPIO_MODE_IT

GPIO_MODE_EVT

RISING_EDGE

FALLING_EDGE

GPIO_OUTPUT_TYPE

GPIO_NUMBER

GPIO pull define

GPIO_NOPULL	No Pull-up or Pull-down activation
GPIO_PULLUP	Pull-up activation
GPIO_PULLDOWN	Pull-down activation

IS_GPIO_PULL

GPIO speed define

GPIO_SPEED_LOW	Low speed
GPIO_SPEED_MEDIUM	Medium speed
GPIO_SPEED_HIGH	High speed

IS_GPIO_SPEED

20 HAL GPIO Extension Driver

20.1 GPIOEx Firmware driver defines

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

20.1.1 GPIOEx

GPIOEx

GPIO Extended Alternate function selection

GPIO_AF0_RTC_50Hz

GPIO_AF0_MCO

GPIO_AF0_TAMPER

GPIO_AF0_SWJ

GPIO_AF0_TRACE

GPIO_AF1_TIM2

GPIO_AF1_TIM15

GPIO_AF1_TIM16

GPIO_AF1_TIM17

GPIO_AF1_EVENTOUT

GPIO_AF2_TIM1

GPIO_AF2_TIM2

GPIO_AF2_TIM3

GPIO_AF2_TIM4

GPIO_AF2_TIM8

GPIO_AF2_TIM15

GPIO_AF2_COMP1

GPIO_AF2_I2C3

GPIO_AF2_TIM20

GPIO_AF3_TSC

GPIO_AF3_TIM8

GPIO_AF3_COMP7

GPIO_AF3_TIM15

GPIO_AF3_I2C3

GPIO_AF3_TIM20

GPIO_AF4_TIM1

GPIO_AF4_TIM8

GPIO_AF4_TIM16

GPIO_AF4_TIM17

GPIO_AF4_I2C1

GPIO_AF4_I2C2

GPIO_AF5_SPI1

GPIO_AF5_SPI2

GPIO_AF5_SPI3

GPIO_AF5_I2S

GPIO_AF5_I2S2ext

GPIO_AF5_TIM8

GPIO_AF5_IR

GPIO_AF5_UART4

GPIO_AF5_UART5

GPIO_AF5_SPI4

GPIO_AF6_SPI2

GPIO_AF6_SPI3

GPIO_AF6_I2S3ext

GPIO_AF6_TIM1

GPIO_AF6_TIM8

GPIO_AF6_IR

GPIO_AF6_TIM20

GPIO_AF7_USART1

GPIO_AF7_USART2

GPIO_AF7_USART3

GPIO_AF7_COMP3

GPIO_AF7_COMP5

GPIO_AF7_COMP6

GPIO_AF7_CAN

GPIO_AF8_COMP1

GPIO_AF8_COMP2

GPIO_AF8_COMP3

GPIO_AF8_COMP4

GPIO_AF8_COMP5

GPIO_AF8_COMP6

GPIO_AF8_I2C3

GPIO_AF9_CAN

GPIO_AF9_TIM1

GPIO_AF9_TIM8

GPIO_AF9_TIM15

GPIO_AF10_TIM2

GPIO_AF10_TIM3

GPIO_AF10_TIM4

GPIO_AF10_TIM8

GPIO_AF10_TIM17

GPIO_AF11_TIM1

GPIO_AF11_TIM8

GPIO_AF12_TIM1

GPIO_AF12_FMC

GPIO_AF12_SDIO

GPIO_AF14_USB

GPIO_AF15_EVENTOUT

IS_GPIO_AF

21 HAL HRTIM Generic Driver

21.1 HRTIM Firmware driver registers structures

21.1.1 HRTIM_InitTypeDef

HRTIM_InitTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t HRTIMInterruptResquests***
- ***uint32_t SyncOptions***
- ***uint32_t SyncInputSource***
- ***uint32_t SyncOutputSource***
- ***uint32_t SyncOutputPolarity***

Field Documentation

- ***uint32_t HRTIM_InitTypeDef::HRTIMInterruptResquests***
  Specifies which interrupts requests must enabled for the HRTIM instance. This parameter can be any combination of ***HRTIM_Common_Interrupt_Enable***
- ***uint32_t HRTIM_InitTypeDef::SyncOptions***
  Specifies how the HRTIM instance handles the external synchronization signals. The HRTIM instance can be configured to act as a slave (waiting for a trigger to be synchronized) or a master (generating a synchronization signal) or both. This parameter can be a combination of ***HRTIM_Synchronization_Options***.
- ***uint32_t HRTIM_InitTypeDef::SyncInputSource***
  Specifies the external synchronization input source (significant only when the HRTIM instance is configured as a slave). This parameter can be a value of ***HRTIM_Synchronization_Input_Source***.
- ***uint32_t HRTIM_InitTypeDef::SyncOutputSource***
  Specifies the source and event to be sent on the external synchronization outputs (significant only when the HRTIM instance is configured as a master). This parameter can be a value of ***HRTIM_Synchronization_Output_Source***
- ***uint32_t HRTIM_InitTypeDef::SyncOutputPolarity***
  Specifies the conditioning of the event to be sent on the external synchronization outputs (significant only when the HRTIM instance is configured as a master). This parameter can be a value of ***HRTIM_Synchronization_Output_Polarity***

21.1.2 HRTIM_TimerParamTypeDef

HRTIM_TimerParamTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t CaptureTrigger1***
- ***uint32_t CaptureTrigger2***
- ***uint32_t InterruptRequests***
- ***uint32_t DMARequests***

- ***uint32_t DMASrcAddress***
- ***uint32_t DMADstAddress***
- ***uint32_t DMASize***

Field Documentation

- ***uint32_t HRTIM_TimerParamTypeDef::CaptureTrigger1***
  Event(s) triggering capture unit 1. When the timer operates in Simple mode, this parameter can be a value of ***HRTIM_External_Event_Channels***. When the timer operates in Waveform mode, this parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***.
- ***uint32_t HRTIM_TimerParamTypeDef::CaptureTrigger2***
  Event(s) triggering capture unit 2. When the timer operates in Simple mode, this parameter can be a value of ***HRTIM_External_Event_Channels***. When the timer operates in Waveform mode, this parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***.
- ***uint32_t HRTIM_TimerParamTypeDef::InterruptRequests***
  Interrupts requests enabled for the timer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMARequests***
  DMA requests enabled for the timer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMASrcAddress***
  Address of the source address of the DMA transfer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMADstAddress***
  Address of the destination address of the DMA transfer.
- ***uint32_t HRTIM_TimerParamTypeDef::DMASize***
  Size of the DMA transfer

21.1.3 __HRTIM_HandleTypeDef

__HRTIM_HandleTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***HRTIM_TypeDef * Instance***
- ***HRTIM_InitTypeDef Init***
- ***HRTIM_TimerParamTypeDef TimerParam***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_HRTIM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdmaMaster***
- ***DMA_HandleTypeDef * hdmaTimerA***
- ***DMA_HandleTypeDef * hdmaTimerB***
- ***DMA_HandleTypeDef * hdmaTimerC***
- ***DMA_HandleTypeDef * hdmaTimerD***
- ***DMA_HandleTypeDef * hdmaTimerE***

Field Documentation

- ***HRTIM_TypeDef* __HRTIM_HandleTypeDef::Instance***
  Register base address
- ***HRTIM_InitTypeDef __HRTIM_HandleTypeDef::Init***
  HRTIM required parameters

- ***HRTIM_TimerParamTypeDef __HRTIM_HandleTypeDef::TimerParam[MAX_HRTIM_TIMER]***
  HRTIM timers - including the master - parameters
- ***HAL_LockTypeDef __HRTIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_HRTIM_StateTypeDef __HRTIM_HandleTypeDef::State***
  HRTIM communication state
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaMaster***
  Master timer DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerA***
  Timer A DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerB***
  Timer B DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerC***
  Timer C DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerD***
  Timer D DMA handle parameters
- ***DMA_HandleTypeDef* __HRTIM_HandleTypeDef::hdmaTimerE***
  Timer E DMA handle parameters

21.1.4 HRTIM_TimeBaseCfgTypeDef

HRTIM_TimeBaseCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Period***
- ***uint32_t RepetitionCounter***
- ***uint32_t PrescalerRatio***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t HRTIM_TimeBaseCfgTypeDef::Period***
  Specifies the timer period. The period value must be above 3 periods of the fHRTIM clock. Maximum value is = 0xFFDF
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::RepetitionCounter***
  Specifies the timer repetition period. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::PrescalerRatio***
  Specifies the timer clock prescaler ratio. This parameter can be any value of ***HRTIM_Prescaler_Ratio***
- ***uint32_t HRTIM_TimeBaseCfgTypeDef::Mode***
  Specifies the counter operating mode. This parameter can be any value of ***HRTIM_Counter_Operating_Mode***

21.1.5 HRTIM_SimpleOCChannelCfgTypeDef

HRTIM_SimpleOCChannelCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Pulse***
- ***uint32_t Polarity***
- ***uint32_t IdleLevel***

Field Documentation

- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Mode***
  Specifies the output compare mode (toggle, active, inactive). This parameter can be any value of of ***HRTIM_Simple_OC_Mode***
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Pulse***
  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::Polarity***
  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_SimpleOCChannelCfgTypeDef::IdleLevel***
  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***

21.1.6 HRTIM_SimplePWMChannelCfgTypeDef

HRTIM_SimplePWMChannelCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Pulse***
- ***uint32_t Polarity***
- ***uint32_t IdleLevel***

Field Documentation

- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::Pulse***
  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::Polarity***
  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_SimplePWMChannelCfgTypeDef::IdleLevel***
  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***

21.1.7 HRTIM_SimpleCaptureChannelCfgTypeDef

HRTIM_SimpleCaptureChannelCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Event***

- ***uint32_t EventPolarity***
- ***uint32_t EventSensitivity***
- ***uint32_t EventFilter***

Field Documentation

- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::Event***
  Specifies the external event triggering the capture. This parameter can be any 'EEVx' value of ***HRTIM_External_Event_Channels***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventPolarity***
  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventSensitivity***
  Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***
- ***uint32_t HRTIM_SimpleCaptureChannelCfgTypeDef::EventFilter***
  Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***

21.1.8 HRTIM_SimpleOnePulseChannelCfgTypeDef

HRTIM_SimpleOnePulseChannelCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Pulse***
- ***uint32_t OutputPolarity***
- ***uint32_t OutputIdleLevel***
- ***uint32_t Event***
- ***uint32_t EventPolarity***
- ***uint32_t EventSensitivity***
- ***uint32_t EventFilter***

Field Documentation

- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::Pulse***
  Specifies the compare value to be loaded into the Compare Register. The compare value must be above or equal to 3 periods of the fHRTIM clock
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::OutputPolarity***
  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::OutputIdleLevel***
  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::Event***
  Specifies the external event triggering the pulse generation. This parameter can be any 'EEVx' value of ***HRTIM_External_Event_Channels***
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventPolarity***
  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***

- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventSensitivity***
 Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***.
- ***uint32_t HRTIM_SimpleOnePulseChannelCfgTypeDef::EventFilter***
 Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***

21.1.9 HRTIM_TimerCfgTypeDef

HRTIM_TimerCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t InterruptRequests***
- ***uint32_t DMARequests***
- ***uint32_t DMASrcAddress***
- ***uint32_t DMADstAddress***
- ***uint32_t DMASize***
- ***uint32_t HalfModeEnable***
- ***uint32_t StartOnSync***
- ***uint32_t ResetOnSync***
- ***uint32_t DACSynchro***
- ***uint32_t PreloadEnable***
- ***uint32_t UpdateGating***
- ***uint32_t BurstMode***
- ***uint32_t RepetitionUpdate***
- ***uint32_t PushPull***
- ***uint32_t FaultEnable***
- ***uint32_t FaultLock***
- ***uint32_t DeadTimeInsertion***
- ***uint32_t DelayedProtectionMode***
- ***uint32_t UpdateTrigger***
- ***uint32_t ResetTrigger***
- ***uint32_t ResetUpdate***

Field Documentation

- ***uint32_t HRTIM_TimerCfgTypeDef::InterruptRequests***
 Relevant for all HRTIM timers, including the master. Specifies which interrupts requests must enabled for the timer. This parameter can be any combination of ***HRTIM_Master_Interrupt_Enable*** or ***HRTIM_Timing_Unit_Interrupt_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::DMARequests***
 Relevant for all HRTIM timers, including the master. Specifies which DMA requests must be enabled for the timer. This parameter can be any combination of ***HRTIM_Master_DMA_Request_Enable*** or ***HRTIM_Timing_Unit_DMA_Request_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::DMASrcAddress***
 Relevant for all HRTIM timers, including the master. Specifies the address of the source address of the DMA transfer
- ***uint32_t HRTIM_TimerCfgTypeDef::DMADstAddress***
 Relevant for all HRTIM timers, including the master. Specifies the address of the destination address of the DMA transfer

- ***uint32_t HRTIM_TimerCfgTypeDef::DMASize***
  Relevant for all HRTIM timers, including the master. Specifies the size of the DMA transfer
- ***uint32_t HRTIM_TimerCfgTypeDef::HalfModeEnable***
  Relevant for all HRTIM timers, including the master. Specifies whether or not hald mode is enabled This parameter can be any value of ***HRTIM_Half_Mode_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::StartOnSync***
  Relevant for all HRTIM timers, including the master. Specifies whether or not timer is reset by a rising edge on the synchronization input (when enabled). This parameter can be any value of ***HRTIM_Start_On_Sync_Input_Event***
- ***uint32_t HRTIM_TimerCfgTypeDef::ResetOnSync***
  Relevant for all HRTIM timers, including the master. Specifies whether or not timer is reset by a rising edge on the synchronization input (when enabled). This parameter can be any value of ***HRTIM_Reset_On_Sync_Input_Event***
- ***uint32_t HRTIM_TimerCfgTypeDef::DACSynchro***
  Relevant for all HRTIM timers, including the master. Indicates whether or not the a DAC synchronization event is generated. This parameter can be any value of ***HRTIM_DAC_Synchronization***
- ***uint32_t HRTIM_TimerCfgTypeDef::PreloadEnable***
  Relevant for all HRTIM timers, including the master. Specifies whether or not register preload is enabled. This parameter can be any value of ***HRTIM_Register_Preload_Enable***
- ***uint32_t HRTIM_TimerCfgTypeDef::UpdateGating***
  Relevant for all HRTIM timers, including the master. Specifies how the update occurs with respect to a burst DMA transaction or update enable inputs (Slave timers only). This parameter can be any value of ***HRTIM_Update_Gating***
- ***uint32_t HRTIM_TimerCfgTypeDef::BurstMode***
  Relevant for all HRTIM timers, including the master. Specifies how the timer behaves during a burst mode operation. This parameter can be any value of ***HRTIM_Timer_Burst_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::RepetitionUpdate***
  Relevant for all HRTIM timers, including the master. Specifies whether or not registers update is triggered by the repetition event. This parameter can be any value of ***HRTIM_Timer_Repetition_Update***
- ***uint32_t HRTIM_TimerCfgTypeDef::PushPull***
  Relevant for Timer A to Timer E. Specifies whether or not the push-pull mode is enabled. This parameter can be any value of ***HRTIM_Timer_Push_Pull_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::FaultEnable***
  Relevant for Timer A to Timer E. Specifies which fault channels are enabled for the timer. This parameter can be a combination of ***HRTIM_Timer_Fault_Enabling***
- ***uint32_t HRTIM_TimerCfgTypeDef::FaultLock***
  Relevant for Timer A to Timer E. Specifies whether or not fault enabling status is write protected. This parameter can be a value of ***HRTIM_Timer_Fault_Lock***
- ***uint32_t HRTIM_TimerCfgTypeDef::DeadTimeInsertion***
  Relevant for Timer A to Timer E. Specifies whether or not dead-time insertion is enabled for the timer. This parameter can be a value of ***HRTIM_Timer_Deadtime_Insertion***
- ***uint32_t HRTIM_TimerCfgTypeDef::DelayedProtectionMode***
  Relevant for Timer A to Timer E. Specifies the delayed protection mode. This parameter can be a value of ***HRTIM_Timer_Delayed_Protection_Mode***
- ***uint32_t HRTIM_TimerCfgTypeDef::UpdateTrigger***
  Relevant for Timer A to Timer E. Specifies source(s) triggering the timer registers update. This parameter can be a combination of ***HRTIM_Timer_Update_Trigger***

- ***uint32_t HRTIM_TimerCfgTypeDef::ResetTrigger***
  Relevant for Timer A to Timer E. Specifies source(s) triggering the timer counter reset. This parameter can be a combination of ***HRTIM_Timer_Reset_Trigger***
- ***uint32_t HRTIM_TimerCfgTypeDef::ResetUpdate***
  Relevant for Timer A to Timer E. Specifies whether or not registers update is triggered when the timer counter is reset. This parameter can be a value of ***HRTIM_Timer_Reset_Update***

21.1.10 HRTIM_CompareCfgTypeDef

HRTIM_CompareCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t CompareValue***
- ***uint32_t AutoDelayedMode***
- ***uint32_t AutoDelayedTimeout***

Field Documentation

- ***uint32_t HRTIM_CompareCfgTypeDef::CompareValue***
  Specifies the compare value of the timer compare unit. The minimum value must be greater than or equal to 3 periods of the fHRTIM clock. The maximum value must be less than or equal to 0xFFFF - 1 periods of the fHRTIM clock
- ***uint32_t HRTIM_CompareCfgTypeDef::AutoDelayedMode***
  Specifies the auto delayed mode for compare unit 2 or 4. This parameter can be a value of ***HRTIM_Compare_Unit_Auto_Delayed_Mode***
- ***uint32_t HRTIM_CompareCfgTypeDef::AutoDelayedTimeout***
  Specifies compare value for timing unit 1 or 3 when auto delayed mode with time out is selected. CompareValue + AutoDelayedTimeout must be less than 0xFFFF

21.1.11 HRTIM_CaptureCfgTypeDef

HRTIM_CaptureCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Trigger***

Field Documentation

- ***uint32_t HRTIM_CaptureCfgTypeDef::Trigger***
  Specifies source(s) triggering the capture. This parameter can be a combination of ***HRTIM_Capture_Unit_Trigger***

21.1.12 HRTIM_OutputCfgTypeDef

HRTIM_OutputCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SetSource***
- ***uint32_t ResetSource***
- ***uint32_t IdleMode***
- ***uint32_t IdleLevel***
- ***uint32_t FaultLevel***
- ***uint32_t ChopperModeEnable***
- ***uint32_t BurstModeEntryDelayed***

Field Documentation

- ***uint32_t HRTIM_OutputCfgTypeDef::Polarity***
  Specifies the output polarity. This parameter can be any value of ***HRTIM_Output_Polarity***
- ***uint32_t HRTIM_OutputCfgTypeDef::SetSource***
  Specifies the event(s) transitioning the output from its inactive level to its active level. This parameter can be a combination of ***HRTIM_Output_Set_Source***
- ***uint32_t HRTIM_OutputCfgTypeDef::ResetSource***
  Specifies the event(s) transitioning the output from its active level to its inactive level. This parameter can be a combination of ***HRTIM_Output_Reset_Source***
- ***uint32_t HRTIM_OutputCfgTypeDef::IdleMode***
  Specifies whether or not the output is affected by a burst mode operation. This parameter can be any value of ***HRTIM_Output_Idle_Mode***
- ***uint32_t HRTIM_OutputCfgTypeDef::IdleLevel***
  Specifies whether the output level is active or inactive when in IDLE state. This parameter can be any value of ***HRTIM_Output_IDLE_Level***
- ***uint32_t HRTIM_OutputCfgTypeDef::FaultLevel***
  Specifies whether the output level is active or inactive when in FAULT state. This parameter can be any value of ***HRTIM_Output_FAULT_Level***
- ***uint32_t HRTIM_OutputCfgTypeDef::ChopperModeEnable***
  Indicates whether or not the chopper mode is enabled This parameter can be any value of ***HRTIM_Output_Chopper_Mode_Enable***
- ***uint32_t HRTIM_OutputCfgTypeDef::BurstModeEntryDelayed***
  Indicates whether or not dead-time is inserted when entering the IDLE state during a burst mode operation. This parameters can be any value of ***HRTIM_Output_Burst_Mode_Entry_Delayed***

21.1.13 HRTIM_TimerEventFilteringCfgTypeDef

HRTIM_TimerEventFilteringCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Filter***
- ***uint32_t Latch***

Field Documentation

- ***uint32_t HRTIM_TimerEventFilteringCfgTypeDef::Filter***
  Specifies the type of event filtering within the timing unit. This parameter can be a value of ***HRTIM_Timer_External_Event_Filter***
- ***uint32_t HRTIM_TimerEventFilteringCfgTypeDef::Latch***
  Specifies whether or not the signal is latched. This parameter can be a value of ***HRTIM_Timer_External_Event_Latch***

21.1.14 HRTIM_DeadTimeCfgTypeDef

HRTIM_DeadTimeCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t RisingValue***
- ***uint32_t RisingSign***
- ***uint32_t RisingLock***
- ***uint32_t RisingSignLock***
- ***uint32_t FallingValue***
- ***uint32_t FallingSign***
- ***uint32_t FallingLock***
- ***uint32_t FallingSignLock***

Field Documentation

- ***uint32_t HRTIM_DeadTimeCfgTypeDef::Prescaler***
  Specifies the Deadtime Prescaler. This parameter can be a value of ***HRTIM_Deadtime_Prescaler_Ratio***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingValue***
  Specifies the Deadtime following a rising edge. This parameter can be a number between 0x0 and 0x1FF
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingSign***
  Specifies whether the deadtime is positive or negative on rising edge. This parameter can be a value of ***HRTIM_Deadtime_Rising_Sign***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingLock***
  Specifies whether or not deadtime rising settings (value and sign) are write protected. This parameter can be a value of ***HRTIM_Deadtime_Rising_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::RisingSignLock***
  Specifies whether or not deadtime rising sign is write protected. This parameter can be a value of ***HRTIM_Deadtime_Rising_Sign_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingValue***
  Specifies the Deadtime following a falling edge. This parameter can be a number between 0x0 and 0x1FF
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingSign***
  Specifies whether the deadtime is positive or negative on falling edge. This parameter can be a value of ***HRTIM_Deadtime_Falling_Sign***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingLock***
  Specifies whether or not deadtime falling settings (value and sign) are write protected. This parameter can be a value of ***HRTIM_Deadtime_Falling_Lock***
- ***uint32_t HRTIM_DeadTimeCfgTypeDef::FallingSignLock***
  Specifies whether or not deadtime falling sign is write protected. This parameter can be a value of ***HRTIM_Deadtime_Falling_Sign_Lock***

21.1.15 HRTIM_ChopperModeCfgTypeDef

HRTIM_ChopperModeCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t CarrierFreq***
- ***uint32_t DutyCycle***
- ***uint32_t StartPulse***

Field Documentation

- ***uint32_t HRTIM_ChopperModeCfgTypeDef::CarrierFreq***
  Specifies the Timer carrier frequency value. This parameter can be a value of ***HRTIM_Chopper_Frequency***
- ***uint32_t HRTIM_ChopperModeCfgTypeDef::DutyCycle***
  Specifies the Timer chopper duty cycle value. This parameter can be a value of ***HRTIM_Chopper_Duty_Cycle***
- ***uint32_t HRTIM_ChopperModeCfgTypeDef::StartPulse***
  Specifies the Timer pulse width value. This parameter can be a value of ***HRTIM_Chopper_Start_Pulse_Width***

21.1.16 HRTIM_EventCfgTypeDef

HRTIM_EventCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t Sensitivity***
- ***uint32_t Filter***
- ***uint32_t FastMode***

Field Documentation

- ***uint32_t HRTIM_EventCfgTypeDef::Source***
  Identifies the source of the external event. This parameter can be a value of ***HRTIM_External_Event_Sources***
- ***uint32_t HRTIM_EventCfgTypeDef::Polarity***
  Specifies the polarity of the external event (in case of level sensitivity). This parameter can be a value of ***HRTIM_External_Event_Polarity***
- ***uint32_t HRTIM_EventCfgTypeDef::Sensitivity***
  Specifies the sensitivity of the external event. This parameter can be a value of ***HRTIM_External_Event_Sensitivity***
- ***uint32_t HRTIM_EventCfgTypeDef::Filter***
  Defines the frequency used to sample the External Event and the length of the digital filter. This parameter can be a value of ***HRTIM_External_Event_Filter***

- ***uint32_t HRTIM_EventCfgTypeDef::FastMode***
  Indicates whether or not low latency mode is enabled for the external event. This parameter can be a value of ***HRTIM_External_Event_Fast_Mode***

21.1.17 HRTIM_FaultCfgTypeDef

HRTIM_FaultCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t Filter***
- ***uint32_t Lock***

Field Documentation

- ***uint32_t HRTIM_FaultCfgTypeDef::Source***
  Identifies the source of the fault. This parameter can be a value of ***HRTIM_Fault_Sources***
- ***uint32_t HRTIM_FaultCfgTypeDef::Polarity***
  Specifies the polarity of the fault event. This parameter can be a value of ***HRTIM_Fault_Polarity***
- ***uint32_t HRTIM_FaultCfgTypeDef::Filter***
  Defines the frequency used to sample the Fault input and the length of the digital filter. This parameter can be a value of ***HRTIM_Fault_Filter***
- ***uint32_t HRTIM_FaultCfgTypeDef::Lock***
  Indicates whether or not fault programming bits are write protected. This parameter can be a value of ***HRTIM_Fault_Lock***

21.1.18 HRTIM_BurstModeCfgTypeDef

HRTIM_BurstModeCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t PreloadEnable***
- ***uint32_t Trigger***
- ***uint32_t IdleDuration***
- ***uint32_t Period***

Field Documentation

- ***uint32_t HRTIM_BurstModeCfgTypeDef::Mode***
  Specifies the burst mode operating mode. This parameter can be a value of ***HRTIM_Burst_Mode_Operating_Mode***

- ***uint32_t HRTIM_BurstModeCfgTypeDef::ClockSource***
  Specifies the burst mode clock source. This parameter can be a value of ***HRTIM_Burst_Mode_Clock_Source***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Prescaler***
  Specifies the burst mode prescaler. This parameter can be a value of ***HRTIM_Burst_Mode_Prescaler***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::PreloadEnable***
  Specifies whether or not preload is enabled for burst mode related registers (HRTIM_BMCMPR and HRTIM_BMPER). This parameter can be a combination of ***HRTIM_Burst_Mode_Register_Preload_Enable***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Trigger***
  Specifies the event(s) triggering the burst operation. This parameter can be a combination of ***HRTIM_Burst_Mode_Trigger***
- ***uint32_t HRTIM_BurstModeCfgTypeDef::IdleDuration***
  Specifies number of periods during which the selected timers are in idle state. This parameter can be a number between 0x0 and 0xFFFF
- ***uint32_t HRTIM_BurstModeCfgTypeDef::Period***
  Specifies burst mode repetition period. This parameter can be a number between 0x1 and 0xFFFF

21.1.19 HRTIM_ADCTriggerCfgTypeDef

HRTIM_ADCTriggerCfgTypeDef is defined in the stm32f3xx_hal_hrtim.h

Data Fields

- ***uint32_t UpdateSource***
- ***uint32_t Trigger***

Field Documentation

- ***uint32_t HRTIM_ADCTriggerCfgTypeDef::UpdateSource***
  Specifies the ADC trigger update source. This parameter can be a combination of ***HRTIM_ADC_Trigger_Update_Source***
- ***uint32_t HRTIM_ADCTriggerCfgTypeDef::Trigger***
  Specifies the event(s) triggering the ADC conversion. This parameter can be a value of ***HRTIM_ADC_Trigger_Event***

21.2 HRTIM Firmware driver API description

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

21.2.1 Simple mode v.s. waveform mode

The HRTIM HAL API is split into 2 categories:

1. Simple functions: these functions allow for using a HRTIM timer as a general purpose timer with high resolution capabilities. HRTIM simple modes are managed through the set of functions named HAL_HRTIM_Simple<Function>. These functions are similar in name and usage to the one defined for the TIM peripheral. When a HRTIM timer

operates in simple mode, only a very limited set of HRTIM features are used. Following simple modes are proposed:
- Output compare mode,
- PWM output mode,
- Input capture mode,
- One pulse mode.

2. Waveform functions: These functions allow taking advantage of the HRTIM flexibility to produce numerous types of control signal. When a HRTIM timer operates in waveform mode, all the HRTIM features are accessible without any restriction. HRTIM waveform modes are managed through the set of functions named HAL_HRTIM_Waveform<Function>

21.2.2 How to use this driver

1. Initialize the HRTIM low level resources by implementing the HAL_HRTIM_MspInit() function:
   a. Enable the HRTIM clock source using __HRTIMx_CLK_ENABLE()
   b. Connect HRTIM pins to MCU I/Os
      - Enable the clock for the HRTIM GPIOs using the following function: __GPIOx_CLK_ENABLE()
      - Configure these GPIO pins in Alternate Function mode using HAL_GPIO_Init()
   c. When using DMA to control data transfer (e.g HAL_HRTIM_SimpleBaseStart_DMA())
      - Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
      - Initialize the DMA handle
      - Associate the initialized DMA handle to the appropriate DMA handle of the HRTIM handle using __HAL_LINKDMA()
      - Initialize the DMA channel using HAL_DMA_Init()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA channel using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
   d. In case of using interrupt mode (e.g HAL_HRTIM_SimpleBaseStart_IT())
      - Configure the priority and enable the NVIC for the concerned HRTIM interrupt using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HRTIM HAL using HAL_HRTIM_Init(). The HRTIM configuration structure (field of the HRTIM handle) specifies which global interrupt of whole HRTIM must be enabled (Burst mode period, System fault, Faults). It also contains the HRTIM external synchronization configuration. HRTIM can act as a master (generating a synchronization signal) or as a slave (waiting for a trigger to be synchronized).
3. Start the high resolution unit using HAL_HRTIM_DLLCalibrationStart(). DLL calibration is executed periodically and compensate for potential voltage and temperature drifts. DLL calibration period is specified by the CalibrationRate argument.
4. HRTIM timers cannot be used until the high resolution unit is ready. This can be checked using HAL_HRTIM_PollForDLLCalibration(): this function returns HAL_OK if DLL calibration is completed or HAL_TIMEOUT if the DLL calibration is still going on when timeout given as argument expires. DLL calibration can also be started in interrupt mode using HAL_HRTIM_DLLCalibrationStart_IT(). In that case an interrupt is generated when the DLL calibration is completed. Note that as DLL calibration is executed on a periodic basis an interrupt will be generated at the end of every DLL calibration operation (worst case: one interrupt every 14 micro seconds !).
5. Configure HRTIM resources shared by all HRTIM timers

a. Burst Mode Controller:
   – HAL_HRTIM_BurstModeConfig(): configures the HRTIM burst mode controller: operating mode (continuous or one-shot mode), clock (source, prescaler) , trigger(s), period, idle duration.
b. External Events Conditionning:
   – HAL_HRTIM_EventConfig(): configures the conditioning of an external event channel: source, polarity, edge-sensitivity. External event can be used as triggers (timer reset, input capture, burst mode, ADC triggers, delayed protection) They can also be used to set or reset timer outputs. Up to 10 event channels are available.
   – HAL_HRTIM_EventPrescalerConfig(): configures the external event sampling clock (used for digital filtering).
c. Fault Conditionning:
   – HAL_HRTIM_FaultConfig(): configures the conditioning of a fault channel: source, polarity, edge-sensitivity. Fault channels are used to disable the outputs in case of an abnormal operation. Up to 5 fault channels are available.
   – HAL_HRTIM_FaultPrescalerConfig(): configures the fault sampling clock (used for digital filtering).
   – HAL_HRTIM_FaultModeCtl(): Enables or disables fault input(s) circuitry. By default all fault inputs are disabled.
d. ADC trigger:
   – HAL_HRTIM_ADCTriggerConfig(): configures the source triggering the update of the ADC trigger register and the ADC trigger. 4 independent triggers are available to start both the regular and the injected sequencers of the 2 ADCs

6. Configure HRTIM timer time base using HAL_HRTIM_TimeBaseConfig(). This function must be called whatever the HRTIM timer operating mode is (simple v.s. waveform). It configures mainly:
   a. The HRTIM timer counter operating mode (continuous v.s. one shot)
   b. The HRTIM timer clock prescaler
   c. The HRTIM timer period
   d. The HRTIM timer repetition counter

If the HRTIM timer operates in simple mode

1. Start or Stop simple timers
   – Simple time base: HAL_HRTIM_SimpleBaseStart(),HAL_HRTIM_SimpleBaseStop(), HAL_HRTIM_SimpleBaseStart_IT(),HAL_HRTIM_SimpleBaseStop_IT(), HAL_HRTIM_SimpleBaseStart_DMA(),HAL_HRTIM_SimpleBaseStop_DMA().
   – Simple output compare: HAL_HRTIM_SimpleOCChannelConfig(), HAL_HRTIM_SimpleOCStart(),HAL_HRTIM_SimpleOCStop(), HAL_HRTIM_SimpleOCStart_IT(),HAL_HRTIM_SimpleOCStop_IT(), HAL_HRTIM_SimpleOCStart_DMA(),HAL_HRTIM_SimpleOCStop_DMA(),
   – Simple PWM output: HAL_HRTIM_SimplePWMChannelConfig(), HAL_HRTIM_SimplePWMStart(),HAL_HRTIM_SimplePWMStop(), HAL_HRTIM_SimplePWMStart_IT(),HAL_HRTIM_SimplePWMStop_IT(), HAL_HRTIM_SimplePWMStart_DMA(),HAL_HRTIM_SimplePWMStop_DMA(),
   – Simple input capture: HAL_HRTIM_SimpleCaptureChannelConfig(), HAL_HRTIM_SimpleCaptureStart(),HAL_HRTIM_SimpleCaptureStop(), HAL_HRTIM_SimpleCaptureStart_IT(),HAL_HRTIM_SimpleCaptureStop_IT(), HAL_HRTIM_SimpleCaptureStart_DMA(),HAL_HRTIM_SimpleCaptureStop_DMA().

- Simple one pulse: HAL_HRTIM_SimpleOnePulseChannelConfig(), HAL_HRTIM_SimpleOnePulseStart(),HAL_HRTIM_SimpleOnePulseStop(), HAL_HRTIM_SimpleOnePulseStart_IT(),HAL_HRTIM_SimpleOnePulseStop_It().

If the HRTIM timer operates in waveform mode

1. Completes waveform timer configuration
   - HAL_HRTIM_WaveformTimerConfig(): configuration of a HRTIM timer operating in wave form mode mainly consists in:
     - Enabling the HRTIM timer interrupts and DMA requests.
     - Enabling the half mode for the HRTIM timer.
     - Defining how the HRTIM timer reacts to external synchronization input.
     - Enabling the push-pull mode for the HRTIM timer.
     - Enabling the fault channels for the HRTIM timer.
     - Enabling the dead-time insertion for the HRTIM timer.
     - Setting the delayed protection mode for the HRTIM timer (source and outputs on which the delayed protection are applied).
     - Specifying the HRTIM timer update and reset triggers.
     - Specifying the HRTIM timer registers update policy (e.g. pre-load enabling).
   - HAL_HRTIM_TimerEventFilteringConfig(): configures external event blanking and windowing circuitry of a HRTIM timer:
     - Blanking: to mask external events during a defined time period a defined time period
     - Windowing, to enable external events only during a defined time period
   - HAL_HRTIM_DeadTimeConfig(): configures the dead-time insertion unit for a HRTIM timer. Allows to generate a couple of complementary signals from a single reference waveform, with programmable delays between active state.
   - HAL_HRTIM_ChopperModeConfig(): configures the parameters of the high-frequency carrier signal added on top of the timing unit output. Chopper mode can be enabled or disabled for each timer output separately (see HAL_HRTIM_WaveformOutputConfig()).
   - HAL_HRTIM_BurstDMAConfig(): configures the burst DMA burst controller. Allows having multiple HRTIM registers updated with a single DMA request. The burst DMA operation is started by calling HAL_HRTIM_BurstDMATransfer().
   - HAL_HRTIM_WaveformCompareConfig():configures the compare unit of a HRTIM timer. This operation consists in setting the compare value and possibly specifying the auto delayed mode for compare units 2 and 4 (allows to have compare events generated relatively to capture events). Note that when auto delayed mode is needed, the capture unit associated to the compare unit must be configured separately.
   - HAL_HRTIM_WaveformCaptureConfig(): configures the capture unit of a HRTIM timer. This operation consists in specifying the source(s) triggering the capture (timer register update event, external event, timer output set/reset event, other HRTIM timer related events).
   - HAL_HRTIM_WaveformOutputConfig(): configuration of a HRTIM timer output mainly consists in:
     - Setting the output polarity (active high or active low),
     - Defining the set/reset crossbar for the output,
     - Specifying the fault level (active or inactive) in IDLE and FAULT states.,
2. Set waveform timer output(s) level
   - HAL_HRTIM_WaveformSetOutputLevel(): forces the output to its active or inactive level. For example, when deadtime insertion is enabled it is necessary to

force the output level by software to have the outputs in a complementary state as soon as the RUN mode is entered.

3. Enable or Disable waveform timer output(s)
   - HAL_HRTIM_WaveformOutputStart(),HAL_HRTIM_WaveformOutputStop().
4. Start or Stop waveform HRTIM timer(s).
   - HAL_HRTIM_WaveformCounterStart(),HAL_HRTIM_WaveformCounterStop(),
   - HAL_HRTIM_WaveformCounterStart_IT(),HAL_HRTIM_WaveformCounterStop_IT(),
   - HAL_HRTIM_WaveformCounterStart()_DMA,HAL_HRTIM_WaveformCounterStop_DMA(),
5. Burst mode controller enabling:
   - HAL_HRTIM_BurstModeCtl(): activates or de-activates the burst mode controller.
6. Some HRTIM operations can be triggered by software:
   - HAL_HRTIM_BurstModeSoftwareTrigger(): calling this function trigs the burst operation.
   - HAL_HRTIM_SoftwareCapture(): calling this function trigs the capture of the HRTIM timer counter.
   - HAL_HRTIM_SoftwareUpdate(): calling this function trigs the update of the pre-loadable registers of the HRTIM timer
   - HAL_HRTIM_SoftwareReset():calling this function resets the HRTIM timer counter.
7. Some functions can be used any time to retrieve HRTIM timer related information
   - HAL_HRTIM_GetCapturedValue(): returns actual value of the capture register of the designated capture unit.
   - HAL_HRTIM_WaveformGetOutputLevel(): returns actual level (ACTIVE/INACTIVE) of the designated timer output.
   - HAL_HRTIM_WaveformGetOutputState():returns actual state (IDLE/RUN/FAULT) of the designated timer output.
   - HAL_HRTIM_GetDelayedProtectionStatus():returns actual level (ACTIVE/INACTIVE) of the designated output when the delayed protection was triggered.
   - HAL_HRTIM_GetBurstStatus(): returns the actual status (ACTIVE/INACTIVE) of the burst mode controller.
   - HAL_HRTIM_GetCurrentPushPullStatus(): when the push-pull mode is enabled for the HRTIM timer (see HAL_HRTIM_WaveformTimerConfig()), the push-pull status indicates on which output the signal is currently active (e.g signal applied on output 1 and output 2 forced inactive or vice versa).
   - HAL_HRTIM_GetIdlePushPullStatus(): when the push-pull mode is enabled for the HRTIM timer (see HAL_HRTIM_WaveformTimerConfig()), the idle push-pull status indicates during which period the delayed protection request occurred (e.g. protection occurred when the output 1 was active and output 2 forced inactive or vice versa).
8. Some functions can be used any time to retrieve actual HRTIM status
   - HAL_HRTIM_GetState(): returns actual HRTIM instance HAL state.

21.2.3 Initialization and Time Base Configuration functions

This section provides functions allowing to:

- Initialize a HRTIM instance
- De-initialize a HRTIM instance
- Initialize the HRTIM MSP
- De-initialize the HRTIM MSP
- Start the high-resolution unit (start DLL calibration)

- Check that the high resolution unit is ready (DLL calibration done)
- Configure the time base unit of a HRTIM timer
- ***HAL_HRTIM_Init()***
- ***HAL_HRTIM_DeInit()***
- ***HAL_HRTIM_MspInit()***
- ***HAL_HRTIM_MspDeInit()***
- ***HAL_HRTIM_DLLCalibrationStart()***
- ***HAL_HRTIM_DLLCalibrationStart_IT()***
- ***HAL_HRTIM_PollForDLLCalibration()***
- ***HAL_HRTIM_TimeBaseConfig()***

21.2.4 Simple time base mode functions

This section provides functions allowing to:

- Start simple time base
- Stop simple time base
- Start simple time base and enable interrupt
- Stop simple time base and disable interrupt
- Start simple time base and enable DMA transfer
- Stop simple time base and disable DMA transfer When a HRTIM timer operates in simple time base mode, the timer counter counts from 0 to the period value.
- ***HAL_HRTIM_SimpleBaseStart()***
- ***HAL_HRTIM_SimpleBaseStop()***
- ***HAL_HRTIM_SimpleBaseStart_IT()***
- ***HAL_HRTIM_SimpleBaseStop_IT()***
- ***HAL_HRTIM_SimpleBaseStart_DMA()***
- ***HAL_HRTIM_SimpleBaseStop_DMA()***

21.2.5 Simple output compare functions

This section provides functions allowing to:

- Configure simple output channel
- Start simple output compare
- Stop simple output compare
- Start simple output compare and enable interrupt
- Stop simple output compare and disable interrupt
- Start simple output compare and enable DMA transfer
- Stop simple output compare and disable DMA transfer When a HRTIM timer operates in simple output compare mode the output level is set to a programmable value when a match is found between the compare register and the counter. Compare unit 1 is automatically associated to output 1 Compare unit 2 is automatically associated to output 2
- ***HAL_HRTIM_SimpleOCChannelConfig()***
- ***HAL_HRTIM_SimpleOCStart()***
- ***HAL_HRTIM_SimpleOCStop()***
- ***HAL_HRTIM_SimpleOCStart_IT()***
- ***HAL_HRTIM_SimpleOCStop_IT()***
- ***HAL_HRTIM_SimpleOCStart_DMA()***
- ***HAL_HRTIM_SimpleOCStop_DMA()***

21.2.6 Simple PWM output functions

This section provides functions allowing to:

- Configure simple PWM output channel
- Start simple PWM output
- Stop simple PWM output
- Start simple PWM output and enable interrupt
- Stop simple PWM output and disable interrupt
- Start simple PWM output and enable DMA transfer
- Stop simple PWM output and disable DMA transfer When a HRTIM timer operates in simple PWM output mode the output level is set to a programmable value when a match is found between the compare register and the counter and reset when the timer period is reached. Duty cycle is determined by the comparison value. Compare unit 1 is automatically associated to output 1 Compare unit 2 is automatically associated to output 2
- ***HAL_HRTIM_SimplePWMChannelConfig()***
- ***HAL_HRTIM_SimplePWMStart()***
- ***HAL_HRTIM_SimplePWMStop()***
- ***HAL_HRTIM_SimplePWMStart_IT()***
- ***HAL_HRTIM_SimplePWMStop_IT()***
- ***HAL_HRTIM_SimplePWMStart_DMA()***
- ***HAL_HRTIM_SimplePWMStop_DMA()***

21.2.7 Simple input capture functions

This section provides functions allowing to:

- Configure simple input capture channel
- Start simple input capture
- Stop simple input capture
- Start simple input capture and enable interrupt
- Stop simple input capture and disable interrupt
- Start simple input capture and enable DMA transfer
- Stop simple input capture and disable DMA transfer When a HRTIM timer operates in simple input capture mode the Capture Register (HRTIM_CPT1/2xR) is used to latch the value of the timer counter counter after a transition detected on a given external event input.
- ***HAL_HRTIM_SimpleCaptureChannelConfig()***
- ***HAL_HRTIM_SimpleCaptureStart()***
- ***HAL_HRTIM_SimpleCaptureStop()***
- ***HAL_HRTIM_SimpleCaptureStart_IT()***
- ***HAL_HRTIM_SimpleCaptureStop_IT()***
- ***HAL_HRTIM_SimpleCaptureStart_DMA()***
- ***HAL_HRTIM_SimpleCaptureStop_DMA()***

21.2.8 Simple one pulse functions

This section provides functions allowing to:

- Configure one pulse channel
- Start one pulse generation

- Stop one pulse generation
- Start one pulse generation and enable interrupt
- Stop one pulse generation and disable interrupt When a HRTIM timer operates in simple one pulse mode the timer counter is started in response to transition detected on a given external event input to generate a pulse with a programmable length after a programmable delay.
- ***HAL_HRTIM_SimpleOnePulseChannelConfig()***
- ***HAL_HRTIM_SimpleOnePulseStart()***
- ***HAL_HRTIM_SimpleOnePulseStop()***
- ***HAL_HRTIM_SimpleOnePulseStart_IT()***
- ***HAL_HRTIM_SimpleOnePulseStop_IT()***

21.2.9 HRTIM configuration functions

This section provides functions allowing to configure the HRTIM resources shared by all the HRTIM timers operating in waveform mode:

- Configure the burst mode controller
- Configure an external event conditionning
- Configure the external events sampling clock
- Configure a fault conditionning
- Enable or disable fault inputs
- Configure the faults sampling clock
- Configure an ADC trigger
- ***HAL_HRTIM_BurstModeConfig()***
- ***HAL_HRTIM_EventConfig()***
- ***HAL_HRTIM_EventPrescalerConfig()***
- ***HAL_HRTIM_FaultConfig()***
- ***HAL_HRTIM_FaultPrescalerConfig()***
- ***HAL_HRTIM_FaultModeCtl()***
- ***HAL_HRTIM_ADCTriggerConfig()***

21.2.10 HRTIM timer configuration and control functions

This section provides functions used to configure and control a HRTIM timer operating in waveform mode:

- Configure HRTIM timer general behavior
- Configure HRTIM timer event filtering
- Configure HRTIM timer deadtime insertion
- Configure HRTIM timer chopper mode
- Configure HRTIM timer burst DMA
- Configure HRTIM timer compare unit
- Configure HRTIM timer capture unit
- Configure HRTIM timer output
- Set HRTIM timer output level
- Enable HRTIM timer output
- Disable HRTIM timer output
- Start HRTIM timer
- Stop HRTIM timer
- Start HRTIM timer and enable interrupt
- Stop HRTIM timer and disable interrupt

- Start HRTIM timer and enable DMA transfer
- Stop HRTIM timer and disable DMA transfer
- Enable or disable the burst mode controller
- Start the burst mode controller (by software)
- Trigger a Capture (by software)
- Update the HRTIM timer preloadable registers (by software)
- Reset the HRTIM timer counter (by software)
- Start a burst DMA transfer
- Enable timer register update
- Disable timer register update
- ***HAL_HRTIM_WaveformTimerConfig()***
- ***HAL_HRTIM_TimerEventFilteringConfig()***
- ***HAL_HRTIM_DeadTimeConfig()***
- ***HAL_HRTIM_ChopperModeConfig()***
- ***HAL_HRTIM_BurstDMAConfig()***
- ***HAL_HRTIM_WaveformCompareConfig()***
- ***HAL_HRTIM_WaveformCaptureConfig()***
- ***HAL_HRTIM_WaveformOutputConfig()***
- ***HAL_HRTIM_WaveformSetOutputLevel()***
- ***HAL_HRTIM_WaveformOutputStart()***
- ***HAL_HRTIM_WaveformOutputStop()***
- ***HAL_HRTIM_WaveformCounterStart()***
- ***HAL_HRTIM_WaveformCounterStop()***
- ***HAL_HRTIM_WaveformCounterStart_IT()***
- ***HAL_HRTIM_WaveformCounterStop_IT()***
- ***HAL_HRTIM_WaveformCounterStart_DMA()***
- ***HAL_HRTIM_WaveformCounterStop_DMA()***
- ***HAL_HRTIM_BurstModeCtl()***
- ***HAL_HRTIM_BurstModeSoftwareTrigger()***
- ***HAL_HRTIM_SoftwareCapture()***
- ***HAL_HRTIM_SoftwareUpdate()***
- ***HAL_HRTIM_SoftwareReset()***
- ***HAL_HRTIM_BurstDMATransfer()***
- ***HAL_HRTIM_UpdateEnable()***
- ***HAL_HRTIM_UpdateDisable()***

21.2.11 Peripheral State functions

This section provides functions used to get HRTIM or HRTIM timer specific information:

- Get HRTIM HAL state
- Get captured value
- Get HRTIM timer output level
- Get HRTIM timer output state
- Get delayed protection status
- Get burst status
- Get current push-pull status
- Get idle push-pull status
- ***HAL_HRTIM_GetState()***
- ***HAL_HRTIM_GetCapturedValue()***
- ***HAL_HRTIM_WaveformGetOutputLevel()***
- ***HAL_HRTIM_WaveformGetOutputState()***

- *HAL_HRTIM_GetDelayedProtectionStatus()*
- *HAL_HRTIM_GetBurstStatus()*
- *HAL_HRTIM_GetCurrentPushPullStatus()*
- *HAL_HRTIM_GetIdlePushPullStatus()*

21.2.12 HAL_HRTIM_Init

Function Name	**HAL_StatusTypeDef HAL_HRTIM_Init (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Initializes a HRTIM instance.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• HAL status

21.2.13 HAL_HRTIM_DeInit

Function Name	**HAL_StatusTypeDef HAL_HRTIM_DeInit (HRTIM_HandleTypeDef * hhrtim)**
Function Description	De-initializes a HRTIM instance.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• HAL status

21.2.14 HAL_HRTIM_MspInit

Function Name	**void HAL_HRTIM_MspInit (HRTIM_HandleTypeDef * hhrtim)**
Function Description	MSP initialization for a HRTIM instance.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.15 HAL_HRTIM_MspDeInit

Function Name	**void HAL_HRTIM_MspDeInit (HRTIM_HandleTypeDef * hhrtim)**
Function Description	MSP de-initialization for a for a HRTIM instance.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.16 HAL_HRTIM_DLLCalibrationStart

Function Name	**HAL_StatusTypeDef HAL_HRTIM_DLLCalibrationStart (HRTIM_HandleTypeDef * hhrtim, uint32_t CalibrationRate)**
Function Description	Starts the DLL calibration.

Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **CalibrationRate:** DLL calibration period This parameter can be one of the following values: HRTIM_SINGLE_CALIBRATION: One shot DLL calibration HRTIM_CALIBRATIONRATE_7300: Periodic DLL calibration. T=7.3 ms HRTIM_CALIBRATIONRATE_910: Periodic DLL calibration. T=910 us HRTIM_CALIBRATIONRATE_114: Periodic DLL calibration. T=114 us HRTIM_CALIBRATIONRATE_14: Periodic DLL calibration. T=14 us
Return values	• HAL status
Notes	• This function locks the HRTIM instance. HRTIM instance is unlocked within the HAL_HRTIM_PollForDLLCalibration function, just before exiting the function.

21.2.17 HAL_HRTIM_DLLCalibrationStart_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_DLLCalibrationStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t CalibrationRate)**
Function Description	Starts the DLL calibration.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **CalibrationRate:** DLL calibration period This parameter can be one of the following values: HRTIM_SINGLE_CALIBRATION: One shot DLL calibration HRTIM_CALIBRATIONRATE_7300: Periodic DLL calibration. T=7.3 ms HRTIM_CALIBRATIONRATE_910: Periodic DLL calibration. T=910 us HRTIM_CALIBRATIONRATE_114: Periodic DLL calibration. T=114 us HRTIM_CALIBRATIONRATE_14: Periodic DLL calibration. T=14 us
Return values	• HAL status
Notes	• This function locks the HRTIM instance. HRTIM instance is unlocked within the IRQ processing function when processing the DLL ready interrupt. • If this function is called for periodic calibration, the DLLRDY interrupt is generated every time the calibration completes which will significantly increases the overall interrupt rate.

21.2.18 HAL_HRTIM_PollForDLLCalibration

Function Name	**HAL_StatusTypeDef HAL_HRTIM_PollForDLLCalibration (HRTIM_HandleTypeDef * hhrtim, uint32_t Timeout)**
Function Description	Polls the DLL calibration ready flag and returns when the flag is set (DLL calibration completed) or upon timeout expiration.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timeout:** Timeout duration in millisecond

Return values	• HAL status

21.2.19 HAL_HRTIM_TimeBaseConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_TimeBaseConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_TimeBaseCfgTypeDef * pTimeBaseCfg)**
Function Description	Configures the time base unit of a timer.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **pTimeBaseCfg:** pointer to the time base configuration structure
Return values	• HAL status
Notes	• This function must be called prior starting the timer • The time-base unit initialization parameters specify: The timer counter operating mode (continuous, one shot), The timer clock prescaler, The timer period , The timer repetition counter.

21.2.20 HAL_HRTIM_SimpleBaseStart

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Starts the counter of a timer operating in simple time base mode.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index. This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• HAL status

21.2.21 HAL_HRTIM_SimpleBaseStop

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Stops the counter of a timer operating in simple time base mode.

Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index. This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• HAL status

21.2.22 HAL_HRTIM_SimpleBaseStart_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Starts the counter of a timer operating in simple time base mode (Timer repetition interrupt is enabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index. This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• HAL status

21.2.23 HAL_HRTIM_SimpleBaseStop_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Stops the counter of a timer operating in simple time base mode (Timer repetition interrupt is disabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index. This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• HAL status

21.2.24 HAL_HRTIM_SimpleBaseStart_DMA

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t**

	SrcAddr, uint32_t DestAddr, uint32_t Length)
Function Description	Starts the counter of a timer operating in simple time base mode (Timer repetition DMA request is enabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index. This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **SrcAddr:** DMA transfer source address • **DestAddr:** DMA transfer destination address • **Length:** The length of data items (data size) to be transferred from source to destination

21.2.25 HAL_HRTIM_SimpleBaseStop_DMA

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleBaseStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Stops the counter of a timer operating in simple time base mode (Timer repetition DMA request is disabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index. This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• HAL status

21.2.26 HAL_HRTIM_SimpleOCChannelConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOCChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel, HRTIM_SimpleOCChannelCfgTypeDef * pSimpleOCChannelCfg)**
Function Description	Configures an output in simple output compare mode.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OCChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1

HRTIM_OUTPUT_TA2: Timer A - Output 2
HRTIM_OUTPUT_TB1: Timer B - Output 1
HRTIM_OUTPUT_TB2: Timer B - Output 2
HRTIM_OUTPUT_TC1: Timer C - Output 1
HRTIM_OUTPUT_TC2: Timer C - Output 2
HRTIM_OUTPUT_TD1: Timer D - Output 1
HRTIM_OUTPUT_TD2: Timer D - Output 2
HRTIM_OUTPUT_TE1: Timer E - Output 1
HRTIM_OUTPUT_TE2: Timer E - Output 2

- **pSimpleOCChannelCfg:** pointer to the simple output compare output configuration structure

Return values
- HAL status

Notes
- When the timer operates in simple output compare mode: Output 1 is implicitly controlled by the compare unit 1 Output 2 is implicitly controlled by the compare unit 2 Output Set/Reset crossbar is set according to the selected output compare mode: Toggle: SETxyR = RSTxyR = CMPy Active: SETxyR = CMPy, RSTxyR = 0 Inactive: SETxy =0, RSTxy = CMPy

21.2.27 HAL_HRTIM_SimpleOCStart

Function Name
HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function Description
Starts the output compare signal generation on the designed timer output.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values
- HAL status

21.2.28 HAL_HRTIM_SimpleOCStop

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)**
Function Description	Stops the output compare signal generation on the designed timer output.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OCChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.29 HAL_HRTIM_SimpleOCStart_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)**
Function Description	Starts the output compare signal generation on the designed timer output (Interrupt is enabled (see note note below)).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OCChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values	• HAL status
Notes	• Interrupt enabling depends on the chosen output compare mode Output toggle: compare match interrupt is enabled Output set active: output set interrupt is enabled Output set inactive: output reset interrupt is enabled

21.2.30 HAL_HRTIM_SimpleOCStop_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)**
Function Description	Stops the output compare signal generation on the designed timer output (Interrupt is disabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OCChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.31 HAL_HRTIM_SimpleOCStart_DMA

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOCStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)**
Function Description	Starts the output compare signal generation on the designed timer output (DMA request is enabled (see note below)).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OCChannel:** Timer output This parameter can be one of the

following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
- **SrcAddr:** DMA transfer source address
- **DestAddr:** DMA transfer destination address
- **Length:** The length of data items (data size) to be transferred from source to destination

Return values
- HAL status

Notes
- DMA request enabling depends on the chosen output compare mode Output toggle: compare match DMA request is enabled Output set active: output set DMA request is enabled Output set inactive: output reset DMA request is enabled

21.2.32 HAL_HRTIM_SimpleOCStop_DMA

Function Name
HAL_StatusTypeDef HAL_HRTIM_SimpleOCStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OCChannel)

Function Description
Stops the output compare signal generation on the designed timer output (DMA request is disabled).

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
- **OCChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values
- HAL status

21.2.33 HAL_HRTIM_SimplePWMChannelConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimplePWMChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel, HRTIM_SimplePWMChannelCfgTypeDef * pSimplePWMChannelCfg)**
Function Description	Configures an output in simple PWM mode.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **PWMChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2 • **pSimplePWMChannelCfg:** pointer to the simple PWM output configuration structure
Return values	• HAL status
Notes	• When the timer operates in simple PWM output mode: Output 1 is implicitly controlled by the compare unit 1 Output 2 is implicitly controlled by the compare unit 2 Output Set/Reset crossbar is set as follows: Output 1: SETx1R = CMP1, RSTx1R = PER Output 2: SETx2R = CMP2, RST2R = PER • When Simple PWM mode is used the registers preload mechanism is enabled (otherwise the behavior is not guaranteed).

21.2.34 HAL_HRTIM_SimplePWMStart

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)**
Function Description	Starts the PWM output signal generation on the designed timer output.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **PWMChannel:** Timer output This parameter can be one of

	the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.35 HAL_HRTIM_SimplePWMStop

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)**
Function Description	Stops the PWM output signal generation on the designed timer output.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **PWMChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.36 HAL_HRTIM_SimplePWMStart_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)**
Function Description	Starts the PWM output signal generation on the designed timer output (The compare interrupt is enabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A

HRTIM_TIMERINDEX_TIMER_B for timer B
HRTIM_TIMERINDEX_TIMER_C for timer C
HRTIM_TIMERINDEX_TIMER_D for timer D
HRTIM_TIMERINDEX_TIMER_E for timer E

- **PWMChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values
- HAL status

21.2.37 HAL_HRTIM_SimplePWMStop_IT

Function Name
HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)

Function Description
Stops the PWM output signal generation on the designed timer output (The compare interrupt is disabled).

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
- **PWMChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values
- HAL status

21.2.38 HAL_HRTIM_SimplePWMStart_DMA

Function Name
HAL_StatusTypeDef HAL_HRTIM_SimplePWMStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t

	Length)
Function Description	Starts the PWM output signal generation on the designed timer output (The compare DMA request is enabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **PWMChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2 • **SrcAddr:** DMA transfer source address • **DestAddr:** DMA transfer destination address • **Length:** The length of data items (data size) to be transferred from source to destination
Return values	• HAL status

21.2.39 HAL_HRTIM_SimplePWMStop_DMA

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimplePWMStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t PWMChannel)**
Function Description	Stops the PWM output signal generation on the designed timer output (The compare DMA request is disabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **PWMChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1

HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values
- HAL status

21.2.40 HAL_HRTIM_SimpleCaptureChannelConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel, HRTIM_SimpleCaptureChannelCfgTypeDef * pSimpleCaptureChannelCfg)**
Function Description	Configures a simple capture.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureChannel:** Capture unit This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2 • **pSimpleCaptureChannelCfg:** pointer to the simple capture configuration structure
Return values	• HAL status
Notes	• When the timer operates in simple capture mode the capture is trigerred by the designated external event and GPIO input is implicitly used as event source. The cature can be triggered by a rising edge, a falling edge or both edges on event channel.

21.2.41 HAL_HRTIM_SimpleCaptureStart

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)**
Function Description	Enables a simple capture on the designed capture unit.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureChannel:** Timer output This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2
Return values	• HAL status

Notes	• The external event triggering the capture is available for all timing units. It can be used directly and is active as soon as the timing unit counter is enabled.

21.2.42 HAL_HRTIM_SimpleCaptureStop

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)**
Function Description	Disables a simple capture on the designed capture unit.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureChannel:** Timer output This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2
Return values	• HAL status

21.2.43 HAL_HRTIM_SimpleCaptureStart_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)**
Function Description	Enables a simple capture on the designed capture unit (Capture interrupt is enabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureChannel:** Timer output This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2
Return values	• HAL status

21.2.44 HAL_HRTIM_SimpleCaptureStop_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)**

Function Description	Disables a simple capture on the designed capture unit (Capture interrupt is disabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureChannel:** Timer output This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2
Return values	• HAL status

21.2.45 HAL_HRTIM_SimpleCaptureStart_DMA

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel, uint32_t SrcAddr, uint32_t DestAddr, uint32_t Length)**
Function Description	Enables a simple capture on the designed capture unit (Capture DMA request is enabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureChannel:** Timer output This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2 • **SrcAddr:** DMA transfer source address • **DestAddr:** DMA transfer destination address • **Length:** The length of data items (data size) to be transferred from source to destination
Return values	• HAL status

21.2.46 HAL_HRTIM_SimpleCaptureStop_DMA

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleCaptureStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureChannel)**
Function Description	Disables a simple capture on the designed capture unit (Capture DMA request is disabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A

HRTIM_TIMERINDEX_TIMER_B for timer B
HRTIM_TIMERINDEX_TIMER_C for timer C
HRTIM_TIMERINDEX_TIMER_D for timer D
HRTIM_TIMERINDEX_TIMER_E for timer E

- **CaptureChannel:** Timer output This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2

Return values
- HAL status

21.2.47 HAL_HRTIM_SimpleOnePulseChannelConfig

Function Name
HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseChannelConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel, HRTIM_SimpleOnePulseChannelCfgTypeDef * pSimpleOnePulseChannelCfg)

Function Description
Configures an output simple one pulse mode.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
- **OnePulseChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pSimpleOnePulseChannelCfg:** pointer to the simple one pulse output configuration structure

Return values
- HAL status

Notes
- When the timer operates in simple one pulse mode: the timer counter is implicitly started by the reset event, the reset of the timer counter is triggered by the designated external event GPIO input is implicitly used as event source, Output 1 is implicitly controlled by the compare unit 1, Output 2 is implicitly controlled by the compare unit 2. Output Set/Reset crossbar is set as follows: Output 1: SETx1R = CMP1, RSTx1R = PER Output 2: SETx2R = CMP2, RST2R = PER
- If HAL_HRTIM_SimpleOnePulseChannelConfig is called for both timer outputs, the reset event related configuration data provided in the second call will override the reset event

related configuration data provided in the first call.

21.2.48 HAL_HRTIM_SimpleOnePulseStart

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStart (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)**
Function Description	Enables the simple one pulse signal generation on the designed output.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OnePulseChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.49 HAL_HRTIM_SimpleOnePulseStop

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStop (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)**
Function Description	Disables the simple one pulse signal generation on the designed output.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OnePulseChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1

	HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.50 HAL_HRTIM_SimpleOnePulseStart_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)**
Function Description	Enables the simple one pulse signal generation on the designed output (The compare interrupt is enabled (pulse start)).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OnePulseChannel:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.51 HAL_HRTIM_SimpleOnePulseStop_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SimpleOnePulseStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t OnePulseChannel)**
Function Description	Disables the simple one pulse signal generation on the designed output (The compare interrupt is disabled).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **OnePulseChannel:** Timer output This parameter can be one

of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2

Return values
- HAL status

21.2.52 HAL_HRTIM_BurstModeConfig

Function Name
HAL_StatusTypeDef HAL_HRTIM_BurstModeConfig (HRTIM_HandleTypeDef * hhrtim, HRTIM_BurstModeCfgTypeDef * pBurstModeCfg)

Function Description
Configures the burst mode feature of the HRTIM.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **pBurstModeCfg:** pointer to the burst mode configuration structure

Return values
- HAL status

Notes
- This function must be called before starting the burst mode controller

21.2.53 HAL_HRTIM_EventConfig

Function Name
HAL_StatusTypeDef HAL_HRTIM_EventConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Event, HRTIM_EventCfgTypeDef * pEventCfg)

Function Description
Configures the conditioning of an external event.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **Event:** external event to configure This parameter can be one of the following values: HRTIM_EVENT_1: External event 1 HRTIM_EVENT_2: External event 2 HRTIM_EVENT_3: External event 3 HRTIM_EVENT_4: External event 4 HRTIM_EVENT_5: External event 5 HRTIM_EVENT_6: External event 6 HRTIM_EVENT_7: External event 7 HRTIM_EVENT_8: External event 8 HRTIM_EVENT_9: External event 9 HRTIM_EVENT_10: External event 10
- **pEventCfg:** pointer to the event conditioning configuration structure

Return values
- HAL status

Notes
- This function must be called before starting the timer

21.2.54 HAL_HRTIM_EventPrescalerConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_EventPrescalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Prescaler)**
Function Description	Configures the external event conditioning block prescaler.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Prescaler:** Prescaler value This parameter can be one of the following values: HRTIM_EVENTPRESCALER_DIV1: fEEVS=fHRTIM HRTIM_EVENTPRESCALER_DIV2: fEEVS=fHRTIM / 2 HRTIM_EVENTPRESCALER_DIV4: fEEVS=fHRTIM / 4 HRTIM_EVENTPRESCALER_DIV8: fEEVS=fHRTIM / 8
Return values	• HAL status
Notes	• This function must be called before starting the timer

21.2.55 HAL_HRTIM_FaultConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_FaultConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Fault, HRTIM_FaultCfgTypeDef * pFaultCfg)**
Function Description	Configures the conditioning of fault input.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Fault:** fault input to configure This parameter can be one of the following values: HRTIM_FAULT_1: Fault input 1 HRTIM_FAULT_2: Fault input 2 HRTIM_FAULT_3: Fault input 3 HRTIM_FAULT_4: Fault input 4 HRTIM_FAULT_5: Fault input 5 • **pFaultCfg:** pointer to the fault conditioning configuration structure
Return values	• HAL status
Notes	• This function must be called before starting the timer and before enabling faults inputs

21.2.56 HAL_HRTIM_FaultPrescalerConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_FaultPrescalerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t Prescaler)**
Function Description	Configures the fault conditioning block prescaler.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Prescaler:** Prescaler value This parameter can be one of the following values: HRTIM_FAULTPRESCALER_DIV1: fFLTS=fHRTIM HRTIM_FAULTPRESCALER_DIV2: fFLTS=fHRTIM / 2 HRTIM_FAULTPRESCALER_DIV4: fFLTS=fHRTIM / 4 HRTIM_FAULTPRESCALER_DIV8: fFLTS=fHRTIM / 8

Return values	• HAL status
Notes	• This function must be called before starting the timer and before enabling faults inputs

21.2.57 HAL_HRTIM_FaultModeCtl

Function Name	**void HAL_HRTIM_FaultModeCtl (HRTIM_HandleTypeDef * hhrtim, uint32_t Faults, uint32_t Enable)**
Function Description	Enables or disables the HRTIMx Fault mode.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Faults:** fault input(s) to enable or disable This parameter can be any combination of the following values: HRTIM_FAULT_1: Fault input 1 HRTIM_FAULT_2: Fault input 2 HRTIM_FAULT_3: Fault input 3 HRTIM_FAULT_4: Fault input 4 HRTIM_FAULT_5: Fault input 5 • **Enable:** Fault(s) enabling This parameter can be one of the following values: HRTIM_FAULTMODECTL_ENABLED: Fault(s) enabled HRTIM_FAULTMODECTL_DISABLED: Fault(s) disabled
Return values	• None

21.2.58 HAL_HRTIM_ADCTriggerConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_ADCTriggerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t ADCTrigger, HRTIM_ADCTriggerCfgTypeDef * pADCTriggerCfg)**
Function Description	Configures both the ADC trigger register update source and the ADC trigger source.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **ADCTrigger:** ADC trigger to configure This parameter can be one of the following values: HRTIM_ADCTRIGGER_1: ADC trigger 1 HRTIM_ADCTRIGGER_2: ADC trigger 2 HRTIM_ADCTRIGGER_3: ADC trigger 3 HRTIM_ADCTRIGGER_4: ADC trigger 4 • **pADCTriggerCfg:** pointer to the ADC trigger configuration structure
Return values	• HAL status
Notes	• This function must be called before starting the timer

21.2.59 HAL_HRTIM_WaveformTimerConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformTimerConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_TimerCfgTypeDef * pTimerCfg)**
Function Description	Configures the general behavior of a timer operating in waveform

	mode.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **pTimerCfg:** pointer to the timer configuration structure
Return values	• HAL status
Notes	• When the timer operates in waveform mode, all the features supported by the HRTIM are available without any limitation. • This function must be called before starting the timer

21.2.60 HAL_HRTIM_TimerEventFilteringConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_TimerEventFilteringConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Event, HRTIM_TimerEventFilteringCfgTypeDef * pTimerEventFilteringCfg)**
Function Description	Configures the event filtering capabilities of a timer (blanking, windowing)
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **Event:** external event for which timer event filtering must be configured This parameter can be one of the following values: HRTIM_EVENT_NONE: Reset timer event filtering configuration HRTIM_EVENT_1: External event 1 HRTIM_EVENT_2: External event 2 HRTIM_EVENT_3: External event 3 HRTIM_EVENT_4: External event 4 HRTIM_EVENT_5: External event 5 HRTIM_EVENT_6: External event 6 HRTIM_EVENT_7: External event 7 HRTIM_EVENT_8: External event 8 HRTIM_EVENT_9: External event 9 HRTIM_EVENT_10: External event 10 • **pTimerEventFilteringCfg:** pointer to the timer event filtering configuration structure
Return values	• HAL status
Notes	• This function must be called before starting the timer

21.2.61 HAL_HRTIM_DeadTimeConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_DeadTimeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_DeadTimeCfgTypeDef * pDeadTimeCfg)**
Function Description	Configures the deadtime insertion feature for a timer.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **pDeadTimeCfg:** pointer to the deadtime insertion configuration structure
Return values	• HAL status
Notes	• This function must be called before starting the timer

21.2.62 HAL_HRTIM_ChopperModeConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_ChopperModeConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, HRTIM_ChopperModeCfgTypeDef * pChopperModeCfg)**
Function Description	Configures the chopper mode feature for a timer.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **pChopperModeCfg:** pointer to the chopper mode configuration structure
Return values	• HAL status
Notes	• This function must be called before configuring the timer output(s)

21.2.63 HAL_HRTIM_BurstDMAConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_BurstDMAConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t RegistersToUpdate)**
Function Description	Configures the burst DMA controller for a timer.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B

HRTIM_TIMERINDEX_TIMER_C for timer C
HRTIM_TIMERINDEX_TIMER_D for timer D
HRTIM_TIMERINDEX_TIMER_E for timer E

- **RegistersToUpdate:** registers to be written by DMA This parameter can be any combination of the following values: HRTIM_BURSTDMA_CR: HRTIM_MCR or HRTIM_TIMxCR HRTIM_BURSTDMA_ICR: HRTIM_MICR or HRTIM_TIMxICR HRTIM_BURSTDMA_DIER: HRTIM_MDIER or HRTIM_TIMxDIER HRTIM_BURSTDMA_CNT: HRTIM_MCNT or HRTIM_TIMxCNT HRTIM_BURSTDMA_PER: HRTIM_MPER or HRTIM_TIMxPER HRTIM_BURSTDMA_REP: HRTIM_MREP or HRTIM_TIMxREP HRTIM_BURSTDMA_CMP1: HRTIM_MCMP1 or HRTIM_TIMxCMP1 HRTIM_BURSTDMA_CMP2: HRTIM_MCMP2 or HRTIM_TIMxCMP2 HRTIM_BURSTDMA_CMP3: HRTIM_MCMP3 or HRTIM_TIMxCMP3 HRTIM_BURSTDMA_CMP4: HRTIM_MCMP4 or HRTIM_TIMxCMP4 HRTIM_BURSTDMA_DTR: HRTIM_TIMxDTR HRTIM_BURSTDMA_SET1R: HRTIM_TIMxSET1R HRTIM_BURSTDMA_RST1R: HRTIM_TIMxRST1R HRTIM_BURSTDMA_SET2R: HRTIM_TIMxSET2R HRTIM_BURSTDMA_RST2R: HRTIM_TIMxRST2R HRTIM_BURSTDMA_EEFR1: HRTIM_TIMxEEFR1 HRTIM_BURSTDMA_EEFR2: HRTIM_TIMxEEFR2 HRTIM_BURSTDMA_RSTR: HRTIM_TIMxRSTR HRTIM_BURSTDMA_CHPR: HRTIM_TIMxCHPR HRTIM_BURSTDMA_OUTR: HRTIM_TIMxOUTR HRTIM_BURSTDMA_FLTR: HRTIM_TIMxFLTR

Return values
- HAL status

Notes
- This function must be called before starting the timer

21.2.64 HAL_HRTIM_WaveformCompareConfig

Function Name
HAL_StatusTypeDef HAL_HRTIM_WaveformCompareConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CompareUnit, HRTIM_CompareCfgTypeDef * pCompareCfg)

Function Description
Configures the compare unit of a timer operating in waveform mode.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
- **CompareUnit:** Compare unit to configure This parameter can be one of the following values: HRTIM_COMPAREUNIT_1: Compare unit 1 HRTIM_COMPAREUNIT_2: Compare unit 2

	HRTIM_COMPAREUNIT_3: Compare unit 3 HRTIM_COMPAREUNIT_4: Compare unit 4 • **pCompareCfg:** pointer to the compare unit configuration structure
Return values	• HAL status
Notes	• When auto delayed mode is required for compare unit 2 or compare unit 4, application has to configure separately the capture unit. Capture unit to configure in that case depends on the compare unit auto delayed mode is applied to (see below): Auto delayed on output compare 2: capture unit 1 must be configured Auto delayed on output compare 4: capture unit 2 must be configured • This function must be called before starting the timer

21.2.65 HAL_HRTIM_WaveformCaptureConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformCaptureConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit, HRTIM_CaptureCfgTypeDef * pCaptureCfg)**
Function Description	Configures the capture unit of a timer operating in waveform mode.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureUnit:** Capture unit to configure This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2 • **pCaptureCfg:** pointer to the compare unit configuration structure
Return values	• HAL status
Notes	• This function must be called before starting the timer

21.2.66 HAL_HRTIM_WaveformOutputConfig

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformOutputConfig (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output, HRTIM_OutputCfgTypeDef * pOutputCfg)**
Function Description	Configures the output of a timer operating in waveform mode.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C

HRTIM_TIMERINDEX_TIMER_D for timer D
HRTIM_TIMERINDEX_TIMER_E for timer E

- **Output:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
- **pOutputCfg:** pointer to the timer output configuration structure

Return values
- HAL status

Notes
- This function must be called before configuring the timer and after configuring the deadtime insertion feature (if required).

21.2.67 HAL_HRTIM_WaveformSetOutputLevel

Function Name
HAL_StatusTypeDef HAL_HRTIM_WaveformSetOutputLevel (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output, uint32_t OutputLevel)

Function Description
Forces the timer output to its active or inactive state.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
- **Output:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
- **OutputLevel:** indicates whether the output is forced to its active or inactive level This parameter can be one of the following values: HRTIM_OUTPUTLEVEL_ACTIVE: output is forced to its active level HRTIM_OUTPUTLEVEL_INACTIVE: output is forced to its inactive level

Return values
- HAL status

Notes
- The 'software set/reset trigger' bit in the output set/reset

registers is automatically reset by hardware

21.2.68 HAL_HRTIM_WaveformOutputStart

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformOutputStart (HRTIM_HandleTypeDef * hhrtim, uint32_t OutputsToStart)**
Function Description	Enables the generation of the waveform signal on the designated output(s) Outputs can be combined (ORed) to allow for simultaneous output enabling.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **OutputsToStart:** Timer output(s) to enable This parameter can be any combination of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.69 HAL_HRTIM_WaveformOutputStop

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformOutputStop (HRTIM_HandleTypeDef * hhrtim, uint32_t OutputsToStop)**
Function Description	Disables the generation of the waveform signal on the designated output(s) Outputs can be combined (ORed) to allow for simultaneous output disabling.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **OutputsToStop:** Timer output(s) to disable This parameter can be any combination of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• HAL status

21.2.70 HAL_HRTIM_WaveformCounterStart

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStart (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Starts the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** Timer counter(s) to start This parameter can be any combination of the following values: HRTIM_TIMERID_MASTER HRTIM_TIMERID_TIMER_A HRTIM_TIMERID_TIMER_B HRTIM_TIMERID_TIMER_C HRTIM_TIMERID_TIMER_D HRTIM_TIMERID_TIMER_E
Return values	• HAL status

21.2.71 HAL_HRTIM_WaveformCounterStop

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStop (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Stops the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values: HRTIM_TIMER_MASTER HRTIM_TIMER_A HRTIM_TIMER_B HRTIM_TIMER_C HRTIM_TIMER_D HRTIM_TIMER_E
Return values	• HAL status
Notes	• The counter of a timer is stopped only if all timer outputs are disabled

21.2.72 HAL_HRTIM_WaveformCounterStart_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStart_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Starts the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** Timer counter(s) to start This parameter can be any combination of the following values: HRTIM_TIMERID_MASTER HRTIM_TIMERID_A HRTIM_TIMERID_B HRTIM_TIMERID_C HRTIM_TIMERID_D HRTIM_TIMERID_E
Return values	• HAL status
Notes	• HRTIM interrupts (e.g. faults interrupts) and interrupts related to the timers to start are enabled within this function.

Interrupts to enable are selected through HAL_HRTIM_WaveformTimerConfig function.

21.2.73 HAL_HRTIM_WaveformCounterStop_IT

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStop_IT (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Stops the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values: HRTIM_TIMER_MASTER HRTIM_TIMER_A HRTIM_TIMER_B HRTIM_TIMER_C HRTIM_TIMER_D HRTIM_TIMER_E
Return values	• HAL status
Notes	• The counter of a timer is stopped only if all timer outputs are disabled • All enabled timer related interrupts are disabled.

21.2.74 HAL_HRTIM_WaveformCounterStart_DMA

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStart_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Starts the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter start.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** Timer counter(s) to start This parameter can be any combination of the following values: HRTIM_TIMER_MASTER HRTIM_TIMER_A HRTIM_TIMER_B HRTIM_TIMER_C HRTIM_TIMER_D HRTIM_TIMER_E
Return values	• HAL status
Notes	• This function enables the dma request(s) mentionned in the timer configuration data structure for every timers to start. • The source memory address, the destination memory address and the size of each DMA transfer are specified at timer configuration time (see HAL_HRTIM_WaveformTimerConfig)

21.2.75 HAL_HRTIM_WaveformCounterStop_DMA

Function Name	**HAL_StatusTypeDef HAL_HRTIM_WaveformCounterStop_DMA (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Stops the counter of the designated timer(s) operating in waveform mode Timers can be combined (ORed) to allow for simultaneous counter stop.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** Timer counter(s) to stop This parameter can be any combination of the following values: HRTIM_TIMER_MASTER HRTIM_TIMER_A HRTIM_TIMER_B HRTIM_TIMER_C HRTIM_TIMER_D HRTIM_TIMER_E
Return values	• HAL status
Notes	• The counter of a timer is stopped only if all timer outputs are disabled • All enabled timer related DMA requests are disabled.

21.2.76 HAL_HRTIM_BurstModeCtl

Function Name	**HAL_StatusTypeDef HAL_HRTIM_BurstModeCtl (HRTIM_HandleTypeDef * hhrtim, uint32_t Enable)**
Function Description	Enables or disables the HRTIM burst mode controller.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Enable:** Burst mode controller enabling This parameter can be one of the following values: HRTIM_BURSTMODECTL_ENABLED: Burst mode enabled HRTIM_BURSTMODECTL_DISABLED: Burst mode disabled
Return values	• HAL status
Notes	• This function must be called after starting the timer(s)

21.2.77 HAL_HRTIM_BurstModeSoftwareTrigger

Function Name	**HAL_StatusTypeDef HAL_HRTIM_BurstModeSoftwareTrigger (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Triggers the burst mode operation.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• HAL status

21.2.78 HAL_HRTIM_SoftwareCapture

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SoftwareCapture (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)**

Function Description	Triggers a software capture on the designed capture unit.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureUnit:** Capture unit to trig This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2
Return values	• HAL status
Notes	• The 'software capture' bit in the capure configuration register is automatically reset by hardware

21.2.79 HAL_HRTIM_SoftwareUpdate

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SoftwareUpdate (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Triggers the update of the registers of one or several timers.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** timers concerned with the software register update This parameter can be any combination of the following values: HRTIM_TIMERUPDATE_MASTER HRTIM_TIMERUPDATE_A HRTIM_TIMERUPDATE_B HRTIM_TIMERUPDATE_C HRTIM_TIMERUPDATE_D HRTIM_TIMERUPDATE_E
Return values	• HAL status
Notes	• The 'software update' bits in the HRTIM conrol register 2 register are automatically reset by hardware

21.2.80 HAL_HRTIM_SoftwareReset

Function Name	**HAL_StatusTypeDef HAL_HRTIM_SoftwareReset (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Triggers the reset of one or several timers.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** timers concerned with the software counter reset This parameter can be any combination of the following values: HRTIM_TIMERRESET_MASTER HRTIM_TIMERRESET_TIMER_A HRTIM_TIMERRESET_TIMER_B HRTIM_TIMERRESET_TIMER_C HRTIM_TIMERRESET_TIMER_D HRTIM_TIMERRESET_TIMER_E
Return values	• HAL status

Notes	• The 'software reset' bits in the HRTIM conrol register 2 are automatically reset by hardware

21.2.81 HAL_HRTIM_BurstDMATransfer

Function Name	**HAL_StatusTypeDef HAL_HRTIM_BurstDMATransfer (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t BurstBufferAddress, uint32_t BurstBufferLength)**
Function Description	Starts a burst DMA operation to update HRTIM control registers content.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **BurstBufferAddress:** address of the buffer the HRTIM control registers content will be updated from. • **BurstBufferLength:** size (in WORDS) of the burst buffer.
Return values	• HAL status
Notes	• The TimerIdx parameter determines the dma channel to be used by the DMA burst controller (see below) HRTIM_TIMERINDEX_MASTER: DMA channel 2 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_A: DMA channel 3 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_B: DMA channel 4 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_C: DMA channel 5 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_D: DMA channel 6 is used by the DMA burst controller HRTIM_TIMERINDEX_TIMER_E: DMA channel 7 is used by the DMA burst controller

21.2.82 HAL_HRTIM_UpdateEnable

Function Name	**HAL_StatusTypeDef HAL_HRTIM_UpdateEnable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Enables the transfer from preload to active registers for one or several timing units (including master timer).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** Timer(s) concerned by the register preload enabling command This parameter can be any combination of the following values: HRTIM_TIMERUPDATE_MASTER HRTIM_TIMERUPDATE_A HRTIM_TIMERUPDATE_B HRTIM_TIMERUPDATE_C HRTIM_TIMERUPDATE_D HRTIM_TIMERUPDATE_E
Return values	• HAL status

21.2.83 HAL_HRTIM_UpdateDisable

Function Name	**HAL_StatusTypeDef HAL_HRTIM_UpdateDisable (HRTIM_HandleTypeDef * hhrtim, uint32_t Timers)**
Function Description	Disables the transfer from preload to active registers for one or several timing units (including master timer).
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **Timers:** Timer(s) concerned by the register preload disabling command This parameter can be any combination of the following values: HRTIM_TIMERUPDATE_MASTER HRTIM_TIMERUPDATE_A HRTIM_TIMERUPDATE_B HRTIM_TIMERUPDATE_C HRTIM_TIMERUPDATE_D HRTIM_TIMERUPDATE_E
Return values	• HAL status

21.2.84 HAL_HRTIM_GetState

Function Name	**HAL_HRTIM_StateTypeDef HAL_HRTIM_GetState (HRTIM_HandleTypeDef * hhrtim)**
Function Description	return the HRTIM HAL state
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• HAL state

21.2.85 HAL_HRTIM_GetCapturedValue

Function Name	**uint32_t HAL_HRTIM_GetCapturedValue (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t CaptureUnit)**
Function Description	Returns actual value of the capture register of the designated capture unit.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **CaptureUnit:** Capture unit to trig This parameter can be one of the following values: HRTIM_CAPTUREUNIT_1: Capture unit 1 HRTIM_CAPTUREUNIT_2: Capture unit 2
Return values	• Captured value

21.2.86 HAL_HRTIM_WaveformGetOutputLevel

Function Name	**uint32_t HAL_HRTIM_WaveformGetOutputLevel (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)**
Function Description	Returns actual level (active or inactive) of the designated output.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **Output:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1 HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• Output level
Notes	• Returned output level is taken before the output stage (chopper, polarity).

21.2.87 HAL_HRTIM_WaveformGetOutputState

Function Name	**uint32_t HAL_HRTIM_WaveformGetOutputState (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)**
Function Description	Returns actual state (RUN, IDLE, FAULT) of the designated output.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **Output:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TE1: Timer E - Output 1

	HRTIM_OUTPUT_TE2: Timer E - Output 2
Return values	• Output state

21.2.88 HAL_HRTIM_GetDelayedProtectionStatus

Function Name	**uint32_t HAL_HRTIM_GetDelayedProtectionStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx, uint32_t Output)**
Function Description	Returns the level (active or inactive) of the designated output when the delayed protection was triggered.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E • **Output:** Timer output This parameter can be one of the following values: HRTIM_OUTPUT_TA1: Timer A - Output 1 HRTIM_OUTPUT_TA2: Timer A - Output 2 HRTIM_OUTPUT_TB1: Timer B - Output 1 HRTIM_OUTPUT_TB2: Timer B - Output 2 HRTIM_OUTPUT_TC1: Timer C - Output 1 HRTIM_OUTPUT_TC2: Timer C - Output 2 HRTIM_OUTPUT_TD1: Timer D - Output 1 HRTIM_OUTPUT_TD2: Timer D - Output 2 HRTIM_OUTPUT_TD1: Timer E - Output 1 HRTIM_OUTPUT_TD2: Timer E - Output 2
Return values	• Delayed protection status

21.2.89 HAL_HRTIM_GetBurstStatus

Function Name	**uint32_t HAL_HRTIM_GetBurstStatus (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Returns the actual status (active or inactive) of the burst mode controller.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• Burst mode controller status

21.2.90 HAL_HRTIM_GetCurrentPushPullStatus

Function Name	**uint32_t HAL_HRTIM_GetCurrentPushPullStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Indicates on which output the signal is currently active (when the push pull mode is enabled).

Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• Burst mode controller status

21.2.91 HAL_HRTIM_GetIdlePushPullStatus

Function Name	**uint32_t HAL_HRTIM_GetIdlePushPullStatus (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Indicates on which output the signal was applied, in push-pull mode, balanced fault mode or delayed idle mode, when the protection was triggered.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• Idle Push Pull Status

21.2.92 HAL_HRTIM_IRQHandler

Function Name	**void HAL_HRTIM_IRQHandler (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	This function handles HRTIM interrupt request.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be any value of HRTIM Timer Index
Return values	• None

21.2.93 HAL_HRTIM_Fault1Callback

Function Name	**void HAL_HRTIM_Fault1Callback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when a fault 1 interrupt occured.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle *
Return values	• None • None

21.2.94 HAL_HRTIM_Fault2Callback

Function Name	**void HAL_HRTIM_Fault2Callback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when a fault 2 interrupt occured.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.95 HAL_HRTIM_Fault3Callback

Function Name	**void HAL_HRTIM_Fault3Callback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when a fault 3 interrupt occured.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.96 HAL_HRTIM_Fault4Callback

Function Name	**void HAL_HRTIM_Fault4Callback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when a fault 4 interrupt occured.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.97 HAL_HRTIM_Fault5Callback

Function Name	**void HAL_HRTIM_Fault5Callback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when a fault 5 interrupt occured.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.98 HAL_HRTIM_SystemFaultCallback

Function Name	**void HAL_HRTIM_SystemFaultCallback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when a system fault interrupt occured.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.99 HAL_HRTIM_DLLCalbrationReadyCallback

Function Name	**void HAL_HRTIM_DLLCalbrationReadyCallback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when the DLL calibration is completed.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.100 HAL_HRTIM_BurstModePeriodCallback

Function Name	**void HAL_HRTIM_BurstModePeriodCallback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when the end of the burst mode period is reached.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.101 HAL_HRTIM_SynchronizationEventCallback

Function Name	**void HAL_HRTIM_SynchronizationEventCallback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when a synchronization input event is received.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.2.102 HAL_HRTIM_RegistersUpdateCallback

Function Name	**void HAL_HRTIM_RegistersUpdateCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when timer registers are updated.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.103 HAL_HRTIM_RepetitionEventCallback

Function Name	**void HAL_HRTIM_RepetitionEventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when timer repetition period has elapsed.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.104 HAL_HRTIM_Compare1EventCallback

Function Name	**void HAL_HRTIM_Compare1EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer counter matches the value programmed in the compare 1 register.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.105 HAL_HRTIM_Compare2EventCallback

Function Name	**void HAL_HRTIM_Compare2EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer counter matches the value programmed in the compare 2 register.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E

Return values	• None

21.2.106 HAL_HRTIM_Compare3EventCallback

Function Name	**void HAL_HRTIM_Compare3EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer counter matches the value programmed in the compare 3 register.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.107 HAL_HRTIM_Compare4EventCallback

Function Name	**void HAL_HRTIM_Compare4EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer counter matches the value programmed in the compare 4 register.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.108 HAL_HRTIM_Capture1EventCallback

Function Name	**void HAL_HRTIM_Capture1EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer x capture 1 event occurs.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D

HRTIM_TIMERINDEX_TIMER_E for timer E

Return values
- None

21.2.109 HAL_HRTIM_Capture2EventCallback

Function Name
void HAL_HRTIM_Capture2EventCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function Description
Callback function invoked when the timer x capture 2 event occurs.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E

Return values
- None

21.2.110 HAL_HRTIM_DelayedProtectionCallback

Function Name
void HAL_HRTIM_DelayedProtectionCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function Description
Callback function invoked when the delayed idle or balanced idle mode is entered.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E

Return values
- None

21.2.111 HAL_HRTIM_CounterResetCallback

Function Name
void HAL_HRTIM_CounterResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)

Function Description
Callback function invoked when the timer x counter reset/roll-over event occurs.

Parameters
- **hhrtim:** pointer to HAL HRTIM handle
- **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D

	HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.112 HAL_HRTIM_Output1SetCallback

Function Name	**void HAL_HRTIM_Output1SetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer x output 1 is set.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.113 HAL_HRTIM_Output1ResetCallback

Function Name	**void HAL_HRTIM_Output1ResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer x output 1 is reset.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.114 HAL_HRTIM_Output2SetCallback

Function Name	**void HAL_HRTIM_Output2SetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer x output 2 is set.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.115 HAL_HRTIM_Output2ResetCallback

Function Name	**void HAL_HRTIM_Output2ResetCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when the timer x output 2 is reset.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.116 HAL_HRTIM_BurstDMATransferCallback

Function Name	**void HAL_HRTIM_BurstDMATransferCallback (HRTIM_HandleTypeDef * hhrtim, uint32_t TimerIdx)**
Function Description	Callback function invoked when a DMA burst transfer is completed.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle • **TimerIdx:** Timer index This parameter can be one of the following values: HRTIM_TIMERINDEX_MASTER for master timer HRTIM_TIMERINDEX_TIMER_A for timer A HRTIM_TIMERINDEX_TIMER_B for timer B HRTIM_TIMERINDEX_TIMER_C for timer C HRTIM_TIMERINDEX_TIMER_D for timer D HRTIM_TIMERINDEX_TIMER_E for timer E
Return values	• None

21.2.117 HAL_HRTIM_ErrorCallback

Function Name	**void HAL_HRTIM_ErrorCallback (HRTIM_HandleTypeDef * hhrtim)**
Function Description	Callback function invoked when a DMA error occurs.
Parameters	• **hhrtim:** pointer to HAL HRTIM handle
Return values	• None

21.3 HRTIM Firmware driver defines

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

21.3.1 HRTIM

HRTIM

HRTIM ADC Trigger

HRTIM_ADCTRIGGER_1	ADC trigger 1 identifier
HRTIM_ADCTRIGGER_2	ADC trigger 2 identifier
HRTIM_ADCTRIGGER_3	ADC trigger 3 identifier
HRTIM_ADCTRIGGER_4	ADC trigger 4 identifier
IS_HRTIM_ADCTRIGGER	

HRTIM ADC Trigger Event

HRTIM_ADCTRIGGEREVENT13_NONE	No ADC trigger event
HRTIM_ADCTRIGGEREVENT13_MASTER_CMP1	ADC Trigger on master compare 1
HRTIM_ADCTRIGGEREVENT13_MASTER_CMP2	ADC Trigger on master compare 2
HRTIM_ADCTRIGGEREVENT13_MASTER_CMP3	ADC Trigger on master compare 3
HRTIM_ADCTRIGGEREVENT13_MASTER_CMP4	ADC Trigger on master compare 4
HRTIM_ADCTRIGGEREVENT13_MASTER_PERIOD	ADC Trigger on master period
HRTIM_ADCTRIGGEREVENT13_EVENT_1	ADC Trigger on external event 1
HRTIM_ADCTRIGGEREVENT13_EVENT_2	ADC Trigger on external event 2
HRTIM_ADCTRIGGEREVENT13_EVENT_3	ADC Trigger on external event 3
HRTIM_ADCTRIGGEREVENT13_EVENT_4	ADC Trigger on external event 4
HRTIM_ADCTRIGGEREVENT13_EVENT_5	ADC Trigger on external event 5
HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP2	ADC Trigger on Timer A compare 2
HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP3	ADC Trigger on Timer A compare 3
HRTIM_ADCTRIGGEREVENT13_TIMERA_CMP4	ADC Trigger on Timer A compare 4
HRTIM_ADCTRIGGEREVENT13_TIMERA_PERIOD	ADC Trigger on Timer A period
HRTIM_ADCTRIGGEREVENT13_TIMERA_RESET	ADC Trigger on Timer A reset
HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP2	ADC Trigger on Timer B compare 2
HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP3	ADC Trigger on Timer B compare 3
HRTIM_ADCTRIGGEREVENT13_TIMERB_CMP4	ADC Trigger on Timer B compare 4
HRTIM_ADCTRIGGEREVENT13_TIMERB_PERIOD	ADC Trigger on Timer B period
HRTIM_ADCTRIGGEREVENT13_TIMERB_RESET	ADC Trigger on Timer B reset
HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP2	ADC Trigger on Timer C compare 2
HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP3	ADC Trigger on Timer C compare 3
HRTIM_ADCTRIGGEREVENT13_TIMERC_CMP4	ADC Trigger on Timer C compare

	4
HRTIM_ADCTRIGGEREVENT13_TIMERC_PERIOD	ADC Trigger on Timer C period
HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP2	ADC Trigger on Timer D compare 2
HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP3	ADC Trigger on Timer D compare 3
HRTIM_ADCTRIGGEREVENT13_TIMERD_CMP4	ADC Trigger on Timer D compare 4
HRTIM_ADCTRIGGEREVENT13_TIMERD_PERIOD	ADC Trigger on Timer D period
HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP2	ADC Trigger on Timer E compare 2
HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP3	ADC Trigger on Timer E compare 3
HRTIM_ADCTRIGGEREVENT13_TIMERE_CMP4	ADC Trigger on Timer E compare 4
HRTIM_ADCTRIGGEREVENT13_TIMERE_PERIOD	ADC Trigger on Timer E period
HRTIM_ADCTRIGGEREVENT24_NONE	No ADC trigger event
HRTIM_ADCTRIGGEREVENT24_MASTER_CMP1	ADC Trigger on master compare 1
HRTIM_ADCTRIGGEREVENT24_MASTER_CMP2	ADC Trigger on master compare 2
HRTIM_ADCTRIGGEREVENT24_MASTER_CMP3	ADC Trigger on master compare 3
HRTIM_ADCTRIGGEREVENT24_MASTER_CMP4	ADC Trigger on master compare 4
HRTIM_ADCTRIGGEREVENT24_MASTER_PERIOD	ADC Trigger on master period
HRTIM_ADCTRIGGEREVENT24_EVENT_6	ADC Trigger on external event 6
HRTIM_ADCTRIGGEREVENT24_EVENT_7	ADC Trigger on external event 7
HRTIM_ADCTRIGGEREVENT24_EVENT_8	ADC Trigger on external event 8
HRTIM_ADCTRIGGEREVENT24_EVENT_9	ADC Trigger on external event 9
HRTIM_ADCTRIGGEREVENT24_EVENT_10	ADC Trigger on external event 10
HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP2	ADC Trigger on Timer A compare 2
HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP3	ADC Trigger on Timer A compare 3
HRTIM_ADCTRIGGEREVENT24_TIMERA_CMP4	ADC Trigger on Timer A compare 4
HRTIM_ADCTRIGGEREVENT24_TIMERA_PERIOD	ADC Trigger on Timer A period
HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP2	ADC Trigger on Timer B compare 2
HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP3	ADC Trigger on Timer B compare 3
HRTIM_ADCTRIGGEREVENT24_TIMERB_CMP4	ADC Trigger on Timer B compare 4

HRTIM_ADCTRIGGEREVENT24_TIMERB_PERIOD	ADC Trigger on Timer B period
HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP2	ADC Trigger on Timer C compare 2
HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP3	ADC Trigger on Timer C compare 3
HRTIM_ADCTRIGGEREVENT24_TIMERC_CMP4	ADC Trigger on Timer C compare 4
HRTIM_ADCTRIGGEREVENT24_TIMERC_PERIOD	ADC Trigger on Timer C period
HRTIM_ADCTRIGGEREVENT24_TIMERC_RESET	ADC Trigger on Timer C reset
HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP2	ADC Trigger on Timer D compare 2
HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP3	ADC Trigger on Timer D compare 3
HRTIM_ADCTRIGGEREVENT24_TIMERD_CMP4	ADC Trigger on Timer D compare 4
HRTIM_ADCTRIGGEREVENT24_TIMERD_PERIOD	ADC Trigger on Timer D period
HRTIM_ADCTRIGGEREVENT24_TIMERD_RESET	ADC Trigger on Timer D reset
HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP2	ADC Trigger on Timer E compare 2
HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP3	ADC Trigger on Timer E compare 3
HRTIM_ADCTRIGGEREVENT24_TIMERE_CMP4	ADC Trigger on Timer E compare 4
HRTIM_ADCTRIGGEREVENT24_TIMERE_RESET	ADC Trigger on Timer E reset

HRTIM ADC Trigger Update Source

HRTIM_ADCTRIGGERUPDATE_MASTER	Master timer
HRTIM_ADCTRIGGERUPDATE_TIMER_A	Timer A
HRTIM_ADCTRIGGERUPDATE_TIMER_B	Timer B
HRTIM_ADCTRIGGERUPDATE_TIMER_C	Timer C
HRTIM_ADCTRIGGERUPDATE_TIMER_D	Timer D
HRTIM_ADCTRIGGERUPDATE_TIMER_E	Timer E

HRTIM Burst DMA Registers Update

HRTIM_BURSTDMA_NONE	No register is updated by Burst DMA accesses
HRTIM_BURSTDMA_CR	MCR or TIMxCR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_ICR	MICR or TIMxICR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_DIER	MDIER or TIMxDIER register is updated by Burst DMA accesses
HRTIM_BURSTDMA_CNT	MCNTR or CNTxCR register is updated by Burst DMA accesses

HRTIM_BURSTDMA_PER	MPER or PERxR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_REP	MREPR or REPxR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_CMP1	MCMP1R or CMP1xR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_CMP2	MCMP2R or CMP2xR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_CMP3	MCMP3R or CMP3xR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_CMP4	MCMP4R or CMP4xR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_DTR	TDxR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_SET1R	SET1R register is updated by Burst DMA accesses
HRTIM_BURSTDMA_RST1R	RST1R register is updated by Burst DMA accesses
HRTIM_BURSTDMA_SET2R	SET2R register is updated by Burst DMA accesses
HRTIM_BURSTDMA_RST2R	RST1R register is updated by Burst DMA accesses
HRTIM_BURSTDMA_EEFR1	EEFxR1 register is updated by Burst DMA accesses
HRTIM_BURSTDMA_EEFR2	EEFxR2 register is updated by Burst DMA accesses
HRTIM_BURSTDMA_RSTR	RSTxR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_CHPR	CHPxR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_OUTR	OUTxR register is updated by Burst DMA accesses
HRTIM_BURSTDMA_FLTR	FLTxR register is updated by Burst DMA accesses

HRTIM Burst Mode Clock Source

HRTIM_BURSTMODECLOCKSOURCE_MASTER	Master timer counter reset/roll-over is used as clock source for the burst mode counter
HRTIM_BURSTMODECLOCKSOURCE_TIMER_A	Timer A counter reset/roll-over is used as clock source for the burst mode counter
HRTIM_BURSTMODECLOCKSOURCE_TIMER_B	Timer B counter reset/roll-over is used as clock source for the burst mode counter
HRTIM_BURSTMODECLOCKSOURCE_TIMER_C	Timer C counter reset/roll-over is used as clock source for the burst mode counter
HRTIM_BURSTMODECLOCKSOURCE_TIMER_D	Timer D counter reset/roll-over is used as clock source for the burst mode counter
HRTIM_BURSTMODECLOCKSOURCE_TIMER_E	Timer E counter reset/roll-over is used as clock source for the burst mode counter
HRTIM_BURSTMODECLOCKSOURCE_TIM16_OC	On-chip Event 1 (BMClk[1]),

	acting as a burst mode counter clock
HRTIM_BURSTMODECLOCKSOURCE_TIM17_OC	On-chip Event 2 (BMClk[2]), acting as a burst mode counter clock
HRTIM_BURSTMODECLOCKSOURCE_TIM7_TRGO	On-chip Event 3 (BMClk[3]), acting as a burst mode counter clock
HRTIM_BURSTMODECLOCKSOURCE_FHRTIM	Prescaled fHRTIM clock is used as clock source for the burst mode counter

HRTIM Burst Mode Control

HRTIM_BURSTMODECTL_DISABLED	Burst mode disabled
HRTIM_BURSTMODECTL_ENABLED	Burst mode enabled

HRTIM Burst Mode Operating Mode

HRTIM_BURSTMODE_SINGLESHOT	Burst mode operates in single shot mode
HRTIM_BURSTMODE_CONTINOUS	Burst mode operates in continuous mode

HRTIM Burst Mode Prescaler

HRTIM_BURSTMODEPRESCALER_DIV1	fBRST = fHRTIM
HRTIM_BURSTMODEPRESCALER_DIV2	fBRST = fHRTIM/2
HRTIM_BURSTMODEPRESCALER_DIV4	fBRST = fHRTIM/4
HRTIM_BURSTMODEPRESCALER_DIV8	fBRST = fHRTIM/8
HRTIM_BURSTMODEPRESCALER_DIV16	fBRST = fHRTIM/16
HRTIM_BURSTMODEPRESCALER_DIV32	fBRST = fHRTIM/32
HRTIM_BURSTMODEPRESCALER_DIV64	fBRST = fHRTIM/64
HRTIM_BURSTMODEPRESCALER_DIV128	fBRST = fHRTIM/128
HRTIM_BURSTMODEPRESCALER_DIV256	fBRST = fHRTIM/256
HRTIM_BURSTMODEPRESCALER_DIV512	fBRST = fHRTIM/512
HRTIM_BURSTMODEPRESCALER_DIV1024	fBRST = fHRTIM/1024
HRTIM_BURSTMODEPRESCALER_DIV2048	fBRST = fHRTIM/2048
HRTIM_BURSTMODEPRESCALER_DIV4096	fBRST = fHRTIM/4096
HRTIM_BURSTMODEPRESCALER_DIV8192	fBRST = fHRTIM/8192
HRTIM_BURSTMODEPRESCALER_DIV16384	fBRST = fHRTIM/16384
HRTIM_BURSTMODEPRESCALER_DIV32768	fBRST = fHRTIM/32768

HRTIM Burst Mode Register Preload Enable

HRIM_BURSTMODEPRELOAD_DISABLED	Preload disabled: the write access is directly done into active registers
HRIM_BURSTMODEPRELOAD_ENABLED	Preload enabled: the write access is done into preload registers

HRTIM Burst Mode Status

HRTIM_BURSTMODESTATUS_NORMAL	Normal operation
HRTIM_BURSTMODESTATUS_ONGOING	Burst operation on-going

HRTIM Burst Mode Trigger

HRTIM_BURSTMODETRIGGER_NONE	No trigger
HRTIM_BURSTMODETRIGGER_MASTER_RESET	Master reset
HRTIM_BURSTMODETRIGGER_MASTER_REPETITION	Master repetition
HRTIM_BURSTMODETRIGGER_MASTER_CMP1	Master compare 1
HRTIM_BURSTMODETRIGGER_MASTER_CMP2	Master compare 2
HRTIM_BURSTMODETRIGGER_MASTER_CMP3	Master compare 3
HRTIM_BURSTMODETRIGGER_MASTER_CMP4	Master compare 4
HRTIM_BURSTMODETRIGGER_TIMERA_RESET	Timer A reset
HRTIM_BURSTMODETRIGGER_TIMERA_REPETITION	Timer A repetition
HRTIM_BURSTMODETRIGGER_TIMERA_CMP1	Timer A compare 1
HRTIM_BURSTMODETRIGGER_TIMERA_CMP2	Timer A compare 2
HRTIM_BURSTMODETRIGGER_TIMERB_RESET	Timer B reset
HRTIM_BURSTMODETRIGGER_TIMERB_REPETITION	Timer B repetition
HRTIM_BURSTMODETRIGGER_TIMERB_CMP1	Timer B compare 1
HRTIM_BURSTMODETRIGGER_TIMERB_CMP2	Timer B compare 2
HRTIM_BURSTMODETRIGGER_TIMERC_RESET	Timer C reset
HRTIM_BURSTMODETRIGGER_TIMERC_REPETITION	Timer C repetition
HRTIM_BURSTMODETRIGGER_TIMERC_CMP1	Timer C compare 1
HRTIM_BURSTMODETRIGGER_TIMERC_CMP2	Timer C compare 2
HRTIM_BURSTMODETRIGGER_TIMERD_RESET	Timer D reset
HRTIM_BURSTMODETRIGGER_TIMERD_REPETITION	Timer D repetition
HRTIM_BURSTMODETRIGGER_TIMERD_CMP1	Timer D compare 1
HRTIM_BURSTMODETRIGGER_TIMERD_CMP2	Timer D compare 2
HRTIM_BURSTMODETRIGGER_TIMERE_RESET	Timer E reset
HRTIM_BURSTMODETRIGGER_TIMERE_REPETITION	Timer E repetition
HRTIM_BURSTMODETRIGGER_TIMERE_CMP1	Timer E compare 1
HRTIM_BURSTMODETRIGGER_TIMERE_CMP2	Timer E compare 2
HRTIM_BURSTMODETRIGGER_TIMERA_EVENT7	Timer A period following External Event 7
HRTIM_BURSTMODETRIGGER_TIMERD_EVENT8	Timer D period following External Event 8
HRTIM_BURSTMODETRIGGER_EVENT_7	External Event 7 (timer A filters applied)
HRTIM_BURSTMODETRIGGER_EVENT_8	External Event 8 (timer D filters applied)

HRTIM_BURSTMODETRIGGER_EVENT_ONCHIP	On-chip Event

HRTIM Capture Unit

HRTIM_CAPTUREUNIT_1	Capture unit 1 identifier
HRTIM_CAPTUREUNIT_2	Capture unit 2 identifier

HRTIM Capture Unit Trigger

HRTIM_CAPTURETRIGGER_NONE	Capture trigger is disabled
HRTIM_CAPTURETRIGGER_UPDATE	The update event triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_1	The External event 1 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_2	The External event 2 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_3	The External event 3 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_4	The External event 4 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_5	The External event 5 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_6	The External event 6 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_7	The External event 7 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_8	The External event 8 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_9	The External event 9 triggers the Capture
HRTIM_CAPTURETRIGGER_EEV_10	The External event 10 triggers the Capture
HRTIM_CAPTURETRIGGER_TA1_SET	Capture is triggered by TA1 output inactive to active transition
HRTIM_CAPTURETRIGGER_TA1_RESET	Capture is triggered by TA1 output active to inactive transition
HRTIM_CAPTURETRIGGER_TIMERA_CMP1	Timer A Compare 1 triggers Capture
HRTIM_CAPTURETRIGGER_TIMERA_CMP2	Timer A Compare 2 triggers Capture
HRTIM_CAPTURETRIGGER_TB1_SET	Capture is triggered by TB1 output inactive to active transition
HRTIM_CAPTURETRIGGER_TB1_RESET	Capture is triggered by TB1 output active to inactive transition
HRTIM_CAPTURETRIGGER_TIMERB_CMP1	Timer B Compare 1 triggers Capture
HRTIM_CAPTURETRIGGER_TIMERB_CMP2	Timer B Compare 2 triggers Capture
HRTIM_CAPTURETRIGGER_TC1_SET	Capture is triggered by TC1 output inactive to active transition
HRTIM_CAPTURETRIGGER_TC1_RESET	Capture is triggered by TC1 output active to inactive transition
HRTIM_CAPTURETRIGGER_TIMERC_CMP1	Timer C Compare 1 triggers Capture
HRTIM_CAPTURETRIGGER_TIMERC_CMP2	Timer C Compare 2 triggers Capture
HRTIM_CAPTURETRIGGER_TD1_SET	Capture is triggered by TD1 output inactive to active transition
HRTIM_CAPTURETRIGGER_TD1_RESET	Capture is triggered by TD1 output active

	to inactive transition
HRTIM_CAPTURETRIGGER_TIMERD_CMP1	Timer D Compare 1 triggers Capture
HRTIM_CAPTURETRIGGER_TIMERD_CMP2	Timer D Compare 2 triggers Capture
HRTIM_CAPTURETRIGGER_TE1_SET	Capture is triggered by TE1 output inactive to active transition
HRTIM_CAPTURETRIGGER_TE1_RESET	Capture is triggered by TE1 output active to inactive transition
HRTIM_CAPTURETRIGGER_TIMERE_CMP1	Timer E Compare 1 triggers Capture
HRTIM_CAPTURETRIGGER_TIMERE_CMP2	Timer E Compare 2 triggers Capture

HRTIM Chopper Duty Cycle

HRTIM_CHOPPER_DUTYCYCLE_0	Only 1st pulse is present
HRTIM_CHOPPER_DUTYCYCLE_125	Duty cycle of the carrier signal is 12.5 %
HRTIM_CHOPPER_DUTYCYCLE_250	Duty cycle of the carrier signal is 25 %
HRTIM_CHOPPER_DUTYCYCLE_375	Duty cycle of the carrier signal is 37.5 %
HRTIM_CHOPPER_DUTYCYCLE_500	Duty cycle of the carrier signal is 50 %
HRTIM_CHOPPER_DUTYCYCLE_625	Duty cycle of the carrier signal is 62.5 %
HRTIM_CHOPPER_DUTYCYCLE_750	Duty cycle of the carrier signal is 75 %
HRTIM_CHOPPER_DUTYCYCLE_875	Duty cycle of the carrier signal is 87.5 %

HRTIM Chopper Frequency

HRTIM_CHOPPER_PRESCALERRATIO_DIV16	fCHPFRQ = fHRTIM / 16
HRTIM_CHOPPER_PRESCALERRATIO_DIV32	fCHPFRQ = fHRTIM / 32
HRTIM_CHOPPER_PRESCALERRATIO_DIV48	fCHPFRQ = fHRTIM / 48
HRTIM_CHOPPER_PRESCALERRATIO_DIV64	fCHPFRQ = fHRTIM / 64
HRTIM_CHOPPER_PRESCALERRATIO_DIV80	fCHPFRQ = fHRTIM / 80
HRTIM_CHOPPER_PRESCALERRATIO_DIV96	fCHPFRQ = fHRTIM / 96
HRTIM_CHOPPER_PRESCALERRATIO_DIV112	fCHPFRQ = fHRTIM / 112
HRTIM_CHOPPER_PRESCALERRATIO_DIV128	fCHPFRQ = fHRTIM / 128
HRTIM_CHOPPER_PRESCALERRATIO_DIV144	fCHPFRQ = fHRTIM / 144
HRTIM_CHOPPER_PRESCALERRATIO_DIV160	fCHPFRQ = fHRTIM / 160
HRTIM_CHOPPER_PRESCALERRATIO_DIV176	fCHPFRQ = fHRTIM / 176
HRTIM_CHOPPER_PRESCALERRATIO_DIV192	fCHPFRQ = fHRTIM / 192
HRTIM_CHOPPER_PRESCALERRATIO_DIV208	fCHPFRQ = fHRTIM / 208
HRTIM_CHOPPER_PRESCALERRATIO_DIV224	fCHPFRQ = fHRTIM / 224
HRTIM_CHOPPER_PRESCALERRATIO_DIV240	fCHPFRQ = fHRTIM / 240
HRTIM_CHOPPER_PRESCALERRATIO_DIV256	fCHPFRQ = fHRTIM / 256

HRTIM Chopper Start Pulse Width

HRTIM_CHOPPER_PULSEWIDTH_16	tSTPW = tHRTIM x 16

HRTIM_CHOPPER_PULSEWIDTH_32	tSTPW = tHRTIM x 32
HRTIM_CHOPPER_PULSEWIDTH_48	tSTPW = tHRTIM x 48
HRTIM_CHOPPER_PULSEWIDTH_64	tSTPW = tHRTIM x 64
HRTIM_CHOPPER_PULSEWIDTH_80	tSTPW = tHRTIM x 80
HRTIM_CHOPPER_PULSEWIDTH_96	tSTPW = tHRTIM x 96
HRTIM_CHOPPER_PULSEWIDTH_112	tSTPW = tHRTIM x 112
HRTIM_CHOPPER_PULSEWIDTH_128	tSTPW = tHRTIM x 128
HRTIM_CHOPPER_PULSEWIDTH_144	tSTPW = tHRTIM x 144
HRTIM_CHOPPER_PULSEWIDTH_160	tSTPW = tHRTIM x 160
HRTIM_CHOPPER_PULSEWIDTH_176	tSTPW = tHRTIM x 176
HRTIM_CHOPPER_PULSEWIDTH_192	tSTPW = tHRTIM x 192
HRTIM_CHOPPER_PULSEWIDTH_208	tSTPW = tHRTIM x 208
HRTIM_CHOPPER_PULSEWIDTH_224	tSTPW = tHRTIM x 224
HRTIM_CHOPPER_PULSEWIDTH_240	tSTPW = tHRTIM x 240
HRTIM_CHOPPER_PULSEWIDTH_256	tSTPW = tHRTIM x 256

HRTIM Common Interrupt Enable

HRTIM_IT_NONE	No interrupt enabled
HRTIM_IT_FLT1	Fault 1 interrupt enable
HRTIM_IT_FLT2	Fault 2 interrupt enable
HRTIM_IT_FLT3	Fault 3 interrupt enable
HRTIM_IT_FLT4	Fault 4 interrupt enable
HRTIM_IT_FLT5	Fault 5 interrupt enable
HRTIM_IT_SYSFLT	System Fault interrupt enable
HRTIM_IT_DLLRDY	DLL ready interrupt enable
HRTIM_IT_BMPER	Burst mode period interrupt enable

HRTIM Common Interrupt Flag

HRTIM_FLAG_FLT1	Fault 1 interrupt flag
HRTIM_FLAG_FLT2	Fault 2 interrupt flag
HRTIM_FLAG_FLT3	Fault 3 interrupt flag
HRTIM_FLAG_FLT4	Fault 4 interrupt flag
HRTIM_FLAG_FLT5	Fault 5 interrupt flag
HRTIM_FLAG_SYSFLT	System Fault interrupt flag
HRTIM_FLAG_DLLRDY	DLL ready interrupt flag
HRTIM_FLAG_BMPER	Burst mode period interrupt flag

HRTIM Compare Unit

HRTIM_COMPAREUNIT_1	Compare unit 1 identifier

HRTIM_COMPAREUNIT_2	Compare unit 2 identifier
HRTIM_COMPAREUNIT_3	Compare unit 3 identifier
HRTIM_COMPAREUNIT_4	Compare unit 4 identifier

HRTIM Compare Unit Auto Delayed Mode

HRTIM_AUTODELAYEDMODE_REGULAR	standard compare mode
HRTIM_AUTODELAYEDMODE_AUTODELAYED_NOTIMEOUT	Compare event generated only if a capture has occurred
HRTIM_AUTODELAYEDMODE_AUTODELAYED_TIMEOUTCMP1	Compare event generated if a capture has occurred or after a Compare 1 match (timeout if capture event is missing)
HRTIM_AUTODELAYEDMODE_AUTODELAYED_TIMEOUTCMP3	Compare event generated if a capture has occurred or after a Compare 3 match (timeout if capture event is missing)

HRTIM Counter Operating Mode

HRTIM_MODE_CONTINUOUS	The timer operates in continuous (free-running) mode
HRTIM_MODE_SINGLESHOT	The timer operates in non retriggerable single-shot mode
HRTIM_MODE_SINGLESHOT_RETRIGGERABLE	The timer operates in retriggerable single-shot mode

HRTIM Current Push Pull Status

HRTIM_PUSHPULL_CURRENTSTATUS_OUTPUT1	Signal applied on output 1 and output 2 forced inactive
HRTIM_PUSHPULL_CURRENTSTATUS_OUTPUT2	Signal applied on output 2 and output 1 forced inactive

HRTIM DAC Synchronization

HRTIM_DACSYNC_NONE	No DAC synchronization event generated
HRTIM_DACSYNC_DACTRIGOUT_1	DAC synchronization event generated on DACTrigOut1 output upon timer update
HRTIM_DACSYNC_DACTRIGOUT_2	DAC synchronization event generated on DACTrigOut2 output upon timer update
HRTIM_DACSYNC_DACTRIGOUT_3	DAC update generated on DACTrigOut3 output upon timer update

HRTIM Deadtime Falling Lock

HRTIM_TIMDEADTIME_FALLINGLOCK_WRITE	Deadtime falling value and sign is writeable
HRTIM_TIMDEADTIME_FALLINGLOCK_READONLY	Deadtime falling value and sign is read-only

HRTIM Deadtime Falling Sign

HRTIM_TIMDEADTIME_FALLINGSIGN_POSITIVE	Positive deadtime on falling edge
HRTIM_TIMDEADTIME_FALLINGSIGN_NEGATIVE	Negative deadtime on falling edge

HRTIM Deadtime Falling Sign Lock

HRTIM_TIMDEADTIME_FALLINGSIGNLOCK_WRITE	Deadtime falling sign is writeable
HRTIM_TIMDEADTIME_FALLINGSIGNLOCK_READONLY	Deadtime falling sign is read-only

HRTIM Deadtime Prescaler Ratio

HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL8	fDTG = fHRTIM * 8
HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL4	fDTG = fHRTIM * 4
HRTIM_TIMDEADTIME_PRESCALERRATIO_MUL2	fDTG = fHRTIM * 2
HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV1	fDTG = fHRTIM
HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV2	fDTG = fHRTIM / 2
HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV4	fDTG = fHRTIM / 4
HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV8	fDTG = fHRTIM / 8
HRTIM_TIMDEADTIME_PRESCALERRATIO_DIV16	fDTG = fHRTIM / 16

HRTIM Deadtime Rising Lock

HRTIM_TIMDEADTIME_RISINGLOCK_WRITE	Deadtime rising value and sign is writeable
HRTIM_TIMDEADTIME_RISINGLOCK_READONLY	Deadtime rising value and sign is read-only

HRTIM Deadtime Rising Sign

HRTIM_TIMDEADTIME_RISINGSIGN_POSITIVE	Positive deadtime on rising edge
HRTIM_TIMDEADTIME_RISINGSIGN_NEGATIVE	Negative deadtime on rising edge

HRTIM Deadtime Rising Sign Lock

HRTIM_TIMDEADTIME_RISINGSIGNLOCK_WRITE	Deadtime rising sign is writeable
HRTIM_TIMDEADTIME_RISINGSIGNLOCK_READONLY	Deadtime rising sign is read-only

HRTIM DLL Calibration Rate

HRTIM_SINGLE_CALIBRATION	Non periodic DLL calibration
HRTIM_CALIBRATIONRATE_7300	Periodic DLL calibration: T = 1048576 * tHRTIM (7.3 ms)
HRTIM_CALIBRATIONRATE_910	Periodic DLL calibration: T = 131072 * tHRTIM (910 ms)

HRTIM_CALIBRATIONRATE_114	Periodic DLL calibration: T = 16384 * tHRTIM (114 ms)
HRTIM_CALIBRATIONRATE_14	Periodic DLL calibration: T = 2048 * tHRTIM (14 ms)

HRTIM Exported Macros

__HAL_HRTIM_RESET_HANDLE_STATE

Description:

- Reset HRTIM handle state.

Parameters:

- __HANDLE__: HRTIM handle.

Return value:

- None:

__HAL_HRTIM_ENABLE

Description:

- Enables or disables the timer counter(s)

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMERS__: timers to enable/disable This parameter can be any combinations of the following values:
  - HRTIM_TIMERID_MASTER: Master timer identifier
  - HRTIM_TIMERID_TIMER_A: Timer A identifier
  - HRTIM_TIMERID_TIMER_B: Timer B identifier
  - HRTIM_TIMERID_TIMER_C: Timer C identifier
  - HRTIM_TIMERID_TIMER_D: Timer D identifier
  - HRTIM_TIMERID_TIMER_E: Timer E identifier

Return value:

- None:

HRTIM_TAOEN_MASK

HRTIM_TBOEN_MASK

HRTIM_TCOEN_MASK

HRTIM_TDOEN_MASK

HRTIM_TEOEN_MASK

__HAL_HRTIM_DISABLE

__HAL_HRTIM_ENABLE_IT

Description:

- Enables or disables the specified HRTIM common interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt enable
  - HRTIM_IT_FLT2: Fault 2 interrupt enable
  - HRTIM_IT_FLT3: Fault 3 interrupt enable
  - HRTIM_IT_FLT4: Fault 4 interrupt enable
  - HRTIM_IT_FLT5: Fault 5 interrupt enable
  - HRTIM_IT_SYSFLT: System Fault interrupt enable
  - HRTIM_IT_DLLRDY: DLL ready interrupt enable
  - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

- None:

__HAL_HRTIM_ENABLE_IT

Description:

- Enables or disables the specified HRTIM common interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt enable
  - HRTIM_IT_FLT2: Fault 2 interrupt enable
  - HRTIM_IT_FLT3: Fault 3 interrupt enable
  - HRTIM_IT_FLT4: Fault 4 interrupt enable
  - HRTIM_IT_FLT5: Fault 5 interrupt enable
  - HRTIM_IT_SYSFLT: System Fault interrupt enable
  - HRTIM_IT_DLLRDY: DLL ready interrupt enable
  - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

- None:

__HAL_HRTIM_DISABLE_IT

__HAL_HRTIM_DISABLE_IT

__HAL_HRTIM_MASTER_ENABLE_IT

Description:

- Enables or disables the specified HRTIM Master timer interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  – HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  – HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  – HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  – HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  – HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  – HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  – HRTIM_MASTER_IT_MUPD: Master update interrupt enable

Return value:

- None:

__HAL_HRTIM_MASTER_DISABLE_IT

__HAL_HRTIM_TIMER_ENABLE_IT

Description:

- Enables or disables the specified HRTIM Timerx interrupts.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  – HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt enable
  – HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt enable
  – HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt enable
  – HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt enable
  – HRTIM_TIM_IT_REP: Timer repetition

interrupt enable
- HRTIM_TIM_IT_UPD: Timer update interrupt enable
- HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt enable
- HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt enable
- HRTIM_TIM_IT_SET1: Timer output 1 set interrupt enable
- HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt enable
- HRTIM_TIM_IT_SET2: Timer output 2 set interrupt enable
- HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt enable
- HRTIM_TIM_IT_RST: Timer reset interrupt enable
- HRTIM_TIM_IT_DLYPRT: Timer delay protection interrupt enable

Return value:

- None:

__HAL_HRTIM_TIMER_DISABLE_IT

__HAL_HRTIM_GET_ITSTATUS

Description:

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

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt enable
  - HRTIM_IT_FLT2: Fault 2 interrupt enable
  - HRTIM_IT_FLT3: Fault 3 enable
  - HRTIM_IT_FLT4: Fault 4 enable
  - HRTIM_IT_FLT5: Fault 5 enable
  - HRTIM_IT_SYSFLT: System Fault interrupt enable
  - HRTIM_IT_DLLRDY: DLL ready interrupt enable
  - HRTIM_IT_BMPER: Burst mode period interrupt enable

Return value:

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

__HAL_HRTIM_MASTER_GET_ITSTATUS

Description:

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

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  - HRTIM_MASTER_IT_MUPD: Master update interrupt enable

Return value:

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

__HAL_HRTIM_TIMER_GET_ITSTATUS

Description:

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

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __INTERRUPT__: specifies the interrupt source to check. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt enable
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt enable
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt enable
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt enable
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt enable
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt enable
  - HRTIM_MASTER_IT_MUPD: Master update interrupt enable

- HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt enable
- HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt enable
- HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt enable
- HRTIM_TIM_IT_CMP4: Timer compare 4 interrupt enable
- HRTIM_TIM_IT_REP: Timer repetition interrupt enable
- HRTIM_TIM_IT_UPD: Timer update interrupt enable
- HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt enable
- HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt enable
- HRTIM_TIM_IT_SET1: Timer output 1 set interrupt enable
- HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt enable
- HRTIM_TIM_IT_SET2: Timer output 2 set interrupt enable
- HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt enable
- HRTIM_TIM_IT_RST: Timer reset interrupt enable
- HRTIM_TIM_IT_DLYPRT: Timer delay protection interrupt enable

Return value:

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

__HAL_HRTIM_CLEAR_IT

Description:

- Clears the specified HRTIM common pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_IT_FLT1: Fault 1 interrupt clear flag
  - HRTIM_IT_FLT2: Fault 2 interrupt clear flag
  - HRTIM_IT_FLT3: Fault 3 clear flag
  - HRTIM_IT_FLT4: Fault 4 clear flag
  - HRTIM_IT_FLT5: Fault 5 clear flag
  - HRTIM_IT_SYSFLT: System Fault interrupt clear flag
  - HRTIM_IT_DLLRDY: DLL ready interrupt clear flag

- HRTIM_IT_BMPER: Burst mode period interrupt clear flag

Return value:

- None:

__HAL_HRTIM_MASTER_CLEAR_IT

Description:

- Clears the specified HRTIM Master pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_MASTER_IT_MCMP1: Master compare 1 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP2: Master compare 2 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP3: Master compare 3 interrupt clear flag
  - HRTIM_MASTER_IT_MCMP4: Master compare 4 interrupt clear flag
  - HRTIM_MASTER_IT_MREP: Master Repetition interrupt clear flag
  - HRTIM_MASTER_IT_SYNC: Synchronization input interrupt clear flag
  - HRTIM_MASTER_IT_MUPD: Master update interrupt clear flag

Return value:

- None:

__HAL_HRTIM_TIMER_CLEAR_IT

Description:

- Clears the specified HRTIM Timerx pending flag.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - HRTIM_TIM_IT_CMP1: Timer compare 1 interrupt clear flag
  - HRTIM_TIM_IT_CMP2: Timer compare 2 interrupt clear flag
  - HRTIM_TIM_IT_CMP3: Timer compare 3 interrupt clear flag
  - HRTIM_TIM_IT_CMP4: Timer

compare 4 interrupt clear flag
- – HRTIM_TIM_IT_REP: Timer repetition interrupt clear flag
- – HRTIM_TIM_IT_UPD: Timer update interrupt clear flag
- – HRTIM_TIM_IT_CPT1: Timer capture 1 interrupt clear flag
- – HRTIM_TIM_IT_CPT2: Timer capture 2 interrupt clear flag
- – HRTIM_TIM_IT_SET1: Timer output 1 set interrupt clear flag
- – HRTIM_TIM_IT_RST1: Timer output 1 reset interrupt clear flag
- – HRTIM_TIM_IT_SET2: Timer output 2 set interrupt clear flag
- – HRTIM_TIM_IT_RST2: Timer output 2 reset interrupt clear flag
- – HRTIM_TIM_IT_RST: Timer reset interrupt clear flag
- – HRTIM_TIM_IT_DLYPRT: Timer output 1 delay protection interrupt clear flag

Return value:

- None:

__HAL_HRTIM_MASTER_ENABLE_DMA

Description:

- Enables or disables the specified HRTIM Master timer DMA requets.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __DMA__: specifies the DMA request to enable or disable. This parameter can be one of the following values:
  - – HRTIM_MASTER_DMA_MCMP1: Master compare 1 DMA resquest enable
  - – HRTIM_MASTER_DMA_MCMP2: Master compare 2 DMA resquest enable
  - – HRTIM_MASTER_DMA_MCMP3: Master compare 3 DMA resquest enable
  - – HRTIM_MASTER_DMA_MCMP4: Master compare 4 DMA resquest enable
  - – HRTIM_MASTER_DMA_MREP: Master Repetition DMA resquest enable
  - – HRTIM_MASTER_DMA_SYNC: Synchronization input DMA resquest enable

- HRTIM_MASTER_DMA_MUPD: Master update DMA resquest enable

Return value:

- None:

__HAL_HRTIM_MASTER_DISABLE_DMA

__HAL_HRTIM_TIMER_ENABLE_DMA

Description:

- Enables or disables the specified HRTIM Timerx DMA requests.

Parameters:

- __HANDLE__: specifies the HRTIM Handle.
- __TIMER__: specified the timing unit (Timer A to E)
- __DMA__: specifies the DMA request to enable or disable. This parameter can be one of the following values:
  - HRTIM_TIM_DMA_CMP1: Timer compare 1 DMA resquest enable
  - HRTIM_TIM_DMA_CMP2: Timer compare 2 DMA resquest enable
  - HRTIM_TIM_DMA_CMP3: Timer compare 3 DMA resquest enable
  - HRTIM_TIM_DMA_CMP4: Timer compare 4 DMA resquest enable
  - HRTIM_TIM_DMA_REP: Timer repetition DMA resquest enable
  - HRTIM_TIM_DMA_UPD: Timer update DMA resquest enable
  - HRTIM_TIM_DMA_CPT1: Timer capture 1 DMA resquest enable
  - HRTIM_TIM_DMA_CPT2: Timer capture 2 DMA resquest enable
  - HRTIM_TIM_DMA_SET1: Timer output 1 set DMA resquest enable
  - HRTIM_TIM_DMA_RST1: Timer output 1 reset DMA resquest enable
  - HRTIM_TIM_DMA_SET2: Timer output 2 set DMA resquest enable
  - HRTIM_TIM_DMA_RST2: Timer output 2 reset DMA resquest enable
  - HRTIM_TIM_DMA_RST: Timer reset DMA resquest enable
  - HRTIM_TIM_DMA_DLYPRT: Timer delay protection DMA resquest enable

Return value:

- None:

__HAL_HRTIM_TIMER_DISABLE_DMA

__HAL_HRTIM_GET_FLAG

__HAL_HRTIM_CLEAR_FLAG

__HAL_HRTIM_MASTER_GET_FLAG

__HAL_HRTIM_MASTER_CLEAR_FLAG

__HAL_HRTIM_TIMER_GET_FLAG

__HAL_HRTIM_TIMER_CLEAR_FLAG

__HAL_HRTIM_SetCounter

Description:

- Sets the HRTIM timer Counter Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E
- __COUNTER__: specifies the Counter Register new value.

Return value:

- None:

__HAL_HRTIM_GetCounter

Description:

- Gets the HRTIM timer Counter Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E

Return value:

- HRTIM: timer Counter Register value

__HAL_HRTIM_SetPeriod

Description:

- Sets the HRTIM timer Period value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E
- __PERIOD__: specifies the Period Register new value.

Return value:

- None:

__HAL_HRTIM_GetPeriod

Description:

- Gets the HRTIM timer Period Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E

Return value:

- timer: Period Register

__HAL_HRTIM_SetClockPrescaler

Description:

- Sets the HRTIM timer clock prescaler value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E
- __PRESCALER__: specifies the clock prescaler new value. This parameter can be one of the following values:
  - HRTIM_PRESCALERRATIO_MUL32: fHRCK: 4.608 GHz - Resolution: 217 ps - Min PWM frequency: 70.3 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_MUL16: fHRCK: 2.304 GHz - Resolution: 434 ps - Min PWM frequency: 35.1 KHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_MUL8: fHRCK: 1.152 GHz - Resolution: 868 ps - Min PWM frequency: 17.6 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_MUL4: fHRCK: 576 MHz - Resolution: 1.73 ns - Min PWM frequency: 8.8 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_MUL2: fHRCK: 288 MHz - Resolution: 3.47 ns - Min PWM frequency: 4.4 kHz (fHRTIM=144MHz)
  - HRTIM_PRESCALERRATIO_DIV1: fHRCK: 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)

- HRTIM_PRESCALERRATIO_DIV2: fHRCK: 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)
- HRTIM_PRESCALERRATIO_DIV4: fHRCK: 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

Return value:

- None:

__HAL_HRTIM_GetClockPrescaler

Description:

- Gets the HRTIM timer clock prescaler value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x5 for master timer
  - 0x0 to 0x4 for timers A to E

Return value:

- timer: clock prescaler value

__HAL_HRTIM_SetCompare

Description:

- Sets the HRTIM timer Compare Register value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x0 to 0x4 for timers A to E
- __COMPAREUNIT__: timer compare unit This parameter can be one of the following values:
  - HRTIM_COMPAREUNIT_1: Compare unit 1
  - HRTIM_COMPAREUNIT_2: Compare unit 2
  - HRTIM_COMPAREUNIT_3: Compare unit 3
  - HRTIM_COMPAREUNIT_4: Compare unit 4
- __COMPARE__: specifies the Compare new value.

Return value:

- None:

__HAL_HRTIM_GetCompare

Description:

- Gets the HRTIM timer Compare Register

value on runtime.

Parameters:

- __HANDLE__: HRTIM Handle.
- __TIMER__: HRTIM timer This parameter can be one of the following values:
  - 0x0 to 0x4 for timers A to E
- __COMPAREUNIT__: timer compare unit This parameter can be one of the following values:
  - HRTIM_COMPAREUNIT_1: Compare unit 1
  - HRTIM_COMPAREUNIT_2: Compare unit 2
  - HRTIM_COMPAREUNIT_3: Compare unit 3
  - HRTIM_COMPAREUNIT_4: Compare unit 4

Return value:

- Compare: value

HRTIM External Event Channels

HRTIM_EVENT_NONE	Undefined event channel
HRTIM_EVENT_1	External event channel 1 identifier
HRTIM_EVENT_2	External event channel 2 identifier
HRTIM_EVENT_3	External event channel 3 identifier
HRTIM_EVENT_4	External event channel 4 identifier
HRTIM_EVENT_5	External event channel 5 identifier
HRTIM_EVENT_6	External event channel 6 identifier
HRTIM_EVENT_7	External event channel 7 identifier
HRTIM_EVENT_8	External event channel 8 identifier
HRTIM_EVENT_9	External event channel 9 identifier
HRTIM_EVENT_10	External event channel 10 identifier

HRTIM External Event Fast Mode

HRTIM_EVENTFASTMODE_DISABLE	External Event is acting asynchronously on outputs (low latency mode)
HRTIM_EVENTFASTMODE_ENABLE	External Event is re-synchronized by the HRTIM logic before acting on outputs

HRTIM External Event Filter

HRTIM_EVENTFILTER_NONE	Filter disabled
HRTIM_EVENTFILTER_1	fSAMPLING= fHRTIM, N=2
HRTIM_EVENTFILTER_2	fSAMPLING= fHRTIM, N=4
HRTIM_EVENTFILTER_3	fSAMPLING= fHRTIM, N=8

HRTIM_EVENTFILTER_4	fSAMPLING= fEEVS/2, N=6
HRTIM_EVENTFILTER_5	fSAMPLING= fEEVS/2, N=8
HRTIM_EVENTFILTER_6	fSAMPLING= fEEVS/4, N=6
HRTIM_EVENTFILTER_7	fSAMPLING= fEEVS/4, N=8
HRTIM_EVENTFILTER_8	fSAMPLING= fEEVS/8, N=6
HRTIM_EVENTFILTER_9	fSAMPLING= fEEVS/8, N=8
HRTIM_EVENTFILTER_10	fSAMPLING= fEEVS/16, N=5
HRTIM_EVENTFILTER_11	fSAMPLING= fEEVS/16, N=6
HRTIM_EVENTFILTER_12	fSAMPLING= fEEVS/16, N=8
HRTIM_EVENTFILTER_13	fSAMPLING= fEEVS/32, N=5
HRTIM_EVENTFILTER_14	fSAMPLING= fEEVS/32, N=6
HRTIM_EVENTFILTER_15	fSAMPLING= fEEVS/32, N=8

HRTIM External Event Polarity

HRTIM_EVENTPOLARITY_HIGH	External event is active high
HRTIM_EVENTPOLARITY_LOW	External event is active low

HRTIM External Event Prescaler

HRTIM_EVENTPRESCALER_DIV1	fEEVS=fHRTIM
HRTIM_EVENTPRESCALER_DIV2	fEEVS=fHRTIM / 2
HRTIM_EVENTPRESCALER_DIV4	fEEVS=fHRTIM / 4
HRTIM_EVENTPRESCALER_DIV8	fEEVS=fHRTIM / 8

HRTIM External Event Sensitivity

HRTIM_EVENTSENSITIVITY_LEVEL	External event is active on level
HRTIM_EVENTSENSITIVITY_RISINGEDGE	External event is active on Rising edge
HRTIM_EVENTSENSITIVITY_FALLINGEDGE	External event is active on Falling edge
HRTIM_EVENTSENSITIVITY_BOTHEDGES	External event is active on Rising and Falling edges

HRTIM External Event Sources

HRTIM_EVENTSRC_1	External event source 1
HRTIM_EVENTSRC_2	External event source 2
HRTIM_EVENTSRC_3	External event source 3
HRTIM_EVENTSRC_4	External event source 4

HRTIM External Fault Prescaler

HRTIM_FAULTPRESCALER_DIV1	fFLTS=fHRTIM
HRTIM_FAULTPRESCALER_DIV2	fFLTS=fHRTIM / 2
HRTIM_FAULTPRESCALER_DIV4	fFLTS=fHRTIM / 4
HRTIM_FAULTPRESCALER_DIV8	fFLTS=fHRTIM / 8

HRTIM Fault Channel

HRTIM_FAULT_1	Fault channel 1 identifier
HRTIM_FAULT_2	Fault channel 2 identifier
HRTIM_FAULT_3	Fault channel 3 identifier
HRTIM_FAULT_4	Fault channel 4 identifier
HRTIM_FAULT_5	Fault channel 5 identifier

HRTIM Fault Filter

HRTIM_FAULTFILTER_NONE	Filter disabled
HRTIM_FAULTFILTER_1	fSAMPLING= fHRTIM, N=2
HRTIM_FAULTFILTER_2	fSAMPLING= fHRTIM, N=4
HRTIM_FAULTFILTER_3	fSAMPLING= fHRTIM, N=8
HRTIM_FAULTFILTER_4	fSAMPLING= fFLTS/2, N=6
HRTIM_FAULTFILTER_5	fSAMPLING= fFLTS/2, N=8
HRTIM_FAULTFILTER_6	fSAMPLING= fFLTS/4, N=6
HRTIM_FAULTFILTER_7	fSAMPLING= fFLTS/4, N=8
HRTIM_FAULTFILTER_8	fSAMPLING= fFLTS/8, N=6
HRTIM_FAULTFILTER_9	fSAMPLING= fFLTS/8, N=8
HRTIM_FAULTFILTER_10	fSAMPLING= fFLTS/16, N=5
HRTIM_FAULTFILTER_11	fSAMPLING= fFLTS/16, N=6
HRTIM_FAULTFILTER_12	fSAMPLING= fFLTS/16, N=8
HRTIM_FAULTFILTER_13	fSAMPLING= fFLTS/32, N=5
HRTIM_FAULTFILTER_14	fSAMPLING= fFLTS/32, N=6
HRTIM_FAULTFILTER_15	fSAMPLING= fFLTS/32, N=8

HRTIM Fault Lock

HRTIM_FAULTLOCK_READWRITE	Fault settings bits are read/write
HRTIM_FAULTLOCK_READONLY	Fault settings bits are read only

HRTIM Fault Mode Control

HRTIM_FAULTMODECTL_DISABLED	Fault channel is disabled
HRTIM_FAULTMODECTL_ENABLED	Fault channel is enabled

IS_HRTIM_FAULTMODECTL

HRTIM Fault Polarity

HRTIM_FAULTPOLARITY_LOW	Fault input is active low
HRTIM_FAULTPOLARITY_HIGH	Fault input is active high

HRTIM Fault Sources

HRTIM_FAULTSOURCE_DIGITALINPUT	Fault input is FLT input pin
HRTIM_FAULTSOURCE_INTERNAL	Fault input is FLT_Int signal (e.g. internal comparator)

HRTIM Half Mode Enable

HRTIM_HALFMODE_DISABLED	Half mode is disabled
HRTIM_HALFMODE_ENABLED	Half mode is enabled

HRTIM Idle Push Pull Status

HRTIM_PUSHPULL_IDLESTATUS_OUTPUT1	Protection occurred when the output 1 was active and output 2 forced inactive
HRTIM_PUSHPULL_IDLESTATUS_OUTPUT2	Protection occurred when the output 2 was active and output 1 forced inactive

HRTIM Master DMA Request Enable

HRTIM_MASTER_DMA_NONE	No DMA request enable
HRTIM_MASTER_DMA_MCMP1	Master compare 1 DMA request enable
HRTIM_MASTER_DMA_MCMP2	Master compare 2 DMA request enable
HRTIM_MASTER_DMA_MCMP3	Master compare 3 DMA request enable
HRTIM_MASTER_DMA_MCMP4	Master compare 4 DMA request enable
HRTIM_MASTER_DMA_MREP	Master Repetition DMA request enable
HRTIM_MASTER_DMA_SYNC	Synchronization input DMA request enable
HRTIM_MASTER_DMA_MUPD	Master update DMA request enable

HRTIM Master Interrupt Enable

HRTIM_MASTER_IT_NONE	No interrupt enabled
HRTIM_MASTER_IT_MCMP1	Master compare 1 interrupt enable
HRTIM_MASTER_IT_MCMP2	Master compare 2 interrupt enable
HRTIM_MASTER_IT_MCMP3	Master compare 3 interrupt enable
HRTIM_MASTER_IT_MCMP4	Master compare 4 interrupt enable
HRTIM_MASTER_IT_MREP	Master Repetition interrupt enable
HRTIM_MASTER_IT_SYNC	Synchronization input interrupt enable
HRTIM_MASTER_IT_MUPD	Master update interrupt enable

HRTIM Master Interrupt Flag

HRTIM_MASTER_FLAG_MCMP1	Master compare 1 interrupt flag
HRTIM_MASTER_FLAG_MCMP2	Master compare 2 interrupt flag
HRTIM_MASTER_FLAG_MCMP3	Master compare 3 interrupt flag
HRTIM_MASTER_FLAG_MCMP4	Master compare 4 interrupt flag
HRTIM_MASTER_FLAG_MREP	Master Repetition interrupt flag
HRTIM_MASTER_FLAG_SYNC	Synchronization input interrupt flag
HRTIM_MASTER_FLAG_MUPD	Master update interrupt flag

HRTIM Max Timer

MAX_HRTIM_TIMER

HRTIM Output Burst Mode Entry Delayed

HRTIM_OUTPUTBURSTMODEENTRY_REGULAR	The programmed Idle state is

	applied immediately to the Output
HRTIM_OUTPUTBURSTMODEENTRY_DELAYED	Deadtime is inserted on output before entering the idle mode

HRTIM Output Chopper Mode Enable

HRTIM_OUTPUTCHOPPERMODE_DISABLED	Output signal is not altered
HRTIM_OUTPUTCHOPPERMODE_ENABLED	Output signal is chopped by a carrier signal

HRTIM Output FAULT Level

HRTIM_OUTPUTFAULTLEVEL_NONE	The output is not affected by the fault input
HRTIM_OUTPUTFAULTLEVEL_ACTIVE	Output at active level when in FAULT state
HRTIM_OUTPUTFAULTLEVEL_INACTIVE	Output at inactive level when in FAULT state
HRTIM_OUTPUTFAULTLEVEL_HIGHZ	Output is tri-stated when in FAULT state

HRTIM Output IDLE Level

HRTIM_OUTPUTIDLELEVEL_INACTIVE	Output at inactive level when in IDLE state
HRTIM_OUTPUTIDLELEVEL_ACTIVE	Output at active level when in IDLE state

HRTIM Output Idle Mode

HRTIM_OUTPUTIDLEMODE_NONE	The output is not affected by the burst mode operation
HRTIM_OUTPUTIDLEMODE_IDLE	The output is in idle state when requested by the burst mode controller

HRTIM Output Level

HRTIM_OUTPUTLEVEL_ACTIVE	Forces the output to its active state
HRTIM_OUTPUTLEVEL_INACTIVE	Forces the output to its inactive state
IS_HRTIM_OUTPUTLEVEL	

HRTIM Output Polarity

HRTIM_OUTPUTPOLARITY_HIGH	Output is acitve HIGH
HRTIM_OUTPUTPOLARITY_LOW	Output is active LOW

HRTIM Output Reset Source

HRTIM_OUTPUTRESET_NONE	Reset the output reset crossbar
HRTIM_OUTPUTRESET_RESYNC	Timer reset event coming solely from software or SYNC input forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMPER	Timer period event forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMCMP1	Timer compare 1 event forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMCMP2	Timer compare 2 event forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMCMP3	Timer compare 3 event forces the output to its inactive state

HRTIM_OUTPUTRESET_TIMCMP4	Timer compare 4 event forces the output to its inactive state
HRTIM_OUTPUTRESET_MASTERPER	The master timer period event forces the output to its inactive state
HRTIM_OUTPUTRESET_MASTERCMP1	Master Timer compare 1 event forces the output to its inactive state
HRTIM_OUTPUTRESET_MASTERCMP2	Master Timer compare 2 event forces the output to its inactive state
HRTIM_OUTPUTRESET_MASTERCMP3	Master Timer compare 3 event forces the output to its inactive state
HRTIM_OUTPUTRESET_MASTERCMP4	Master Timer compare 4 event forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_1	Timer event 1 forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_2	Timer event 2 forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_3	Timer event 3 forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_4	Timer event 4 forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_5	Timer event 5 forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_6	Timer event 6 forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_7	Timer event 7 forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_8	Timer event 8 forces the output to its inactive state
HRTIM_OUTPUTRESET_TIMEV_9	Timer event 9 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_1	External event 1 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_2	External event 2 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_3	External event 3 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_4	External event 4 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_5	External event 5 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_6	External event 6 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_7	External event 7 forces the output to its inactive state

HRTIM_OUTPUTRESET_EEV_8	External event 8 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_9	External event 9 forces the output to its inactive state
HRTIM_OUTPUTRESET_EEV_10	External event 10 forces the output to its inactive state
HRTIM_OUTPUTRESET_UPDATE	Timer register update event forces the output to its inactive state

HRTIM Output Set Source

HRTIM_OUTPUTSET_NONE	Reset the output set crossbar
HRTIM_OUTPUTSET_RESYNC	Timer reset event coming solely from software or SYNC input forces the output to its active state
HRTIM_OUTPUTSET_TIMPER	Timer period event forces the output to its active state
HRTIM_OUTPUTSET_TIMCMP1	Timer compare 1 event forces the output to its active state
HRTIM_OUTPUTSET_TIMCMP2	Timer compare 2 event forces the output to its active state
HRTIM_OUTPUTSET_TIMCMP3	Timer compare 3 event forces the output to its active state
HRTIM_OUTPUTSET_TIMCMP4	Timer compare 4 event forces the output to its active state
HRTIM_OUTPUTSET_MASTERPER	The master timer period event forces the output to its active state
HRTIM_OUTPUTSET_MASTERCMP1	Master Timer compare 1 event forces the output to its active state
HRTIM_OUTPUTSET_MASTERCMP2	Master Timer compare 2 event forces the output to its active state
HRTIM_OUTPUTSET_MASTERCMP3	Master Timer compare 3 event forces the output to its active state
HRTIM_OUTPUTSET_MASTERCMP4	Master Timer compare 4 event forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_1	Timer event 1 forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_2	Timer event 2 forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_3	Timer event 3 forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_4	Timer event 4 forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_5	Timer event 5 forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_6	Timer event 6 forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_7	Timer event 7 forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_8	Timer event 8 forces the output to its active state
HRTIM_OUTPUTSET_TIMEV_9	Timer event 9 forces the output to its active state

HRTIM_OUTPUTSET_EEV_1	External event 1 forces the output to its active state
HRTIM_OUTPUTSET_EEV_2	External event 2 forces the output to its active state
HRTIM_OUTPUTSET_EEV_3	External event 3 forces the output to its active state
HRTIM_OUTPUTSET_EEV_4	External event 4 forces the output to its active state
HRTIM_OUTPUTSET_EEV_5	External event 5 forces the output to its active state
HRTIM_OUTPUTSET_EEV_6	External event 6 forces the output to its active state
HRTIM_OUTPUTSET_EEV_7	External event 7 forces the output to its active state
HRTIM_OUTPUTSET_EEV_8	External event 8 forces the output to its active state
HRTIM_OUTPUTSET_EEV_9	External event 9 forces the output to its active state
HRTIM_OUTPUTSET_EEV_10	External event 10 forces the output to its active state
HRTIM_OUTPUTSET_UPDATE	Timer register update event forces the output to its active state

HRTIM Output State

HRTIM_OUTPUTSTATE_IDLE	Main operating mode, where the output can take the active or inactive level as programmed in the crossbar unit
HRTIM_OUTPUTSTATE_RUN	Default operating state (e.g. after an HRTIM reset, when the outputs are disabled by software or during a burst mode operation
HRTIM_OUTPUTSTATE_FAULT	Safety state, entered in case of a shut-down request on FAULTx inputs

HRTIM Prescaler Ratio

HRTIM_PRESCALERRATIO_MUL32	fHRCK: fHRTIM x 32 = 4.608 GHz - Resolution: 217 ps - Min PWM frequency: 70.3 kHz (fHRTIM=144MHz)
HRTIM_PRESCALERRATIO_MUL16	fHRCK: fHRTIM x 16 = 2.304 GHz - Resolution: 434 ps - Min PWM frequency: 35.1 KHz (fHRTIM=144MHz)
HRTIM_PRESCALERRATIO_MUL8	fHRCK: fHRTIM x 8 = 1.152 GHz - Resolution: 868 ps - Min PWM frequency: 17.6 kHz (fHRTIM=144MHz)
HRTIM_PRESCALERRATIO_MUL4	fHRCK: fHRTIM x 4 = 576 MHz - Resolution: 1.73 ns - Min PWM frequency: 8.8 kHz (fHRTIM=144MHz)
HRTIM_PRESCALERRATIO_MUL2	fHRCK: fHRTIM x 2 = 288 MHz - Resolution: 3.47

	ns - Min PWM frequency: 4.4 kHz (fHRTIM=144MHz)
HRTIM_PRESCALERRATIO_DIV1	fHRCK: fHRTIM = 144 MHz - Resolution: 6.95 ns - Min PWM frequency: 2.2 kHz (fHRTIM=144MHz)
HRTIM_PRESCALERRATIO_DIV2	fHRCK: fHRTIM / 2 = 72 MHz - Resolution: 13.88 ns- Min PWM frequency: 1.1 kHz (fHRTIM=144MHz)
HRTIM_PRESCALERRATIO_DIV4	fHRCK: fHRTIM / 4 = 36 MHz - Resolution: 27.7 ns- Min PWM frequency: 550Hz (fHRTIM=144MHz)

HRTIM Private Define

HRTIM_FLTR_FLTxEN

HRTIM_TIMCR_TIMUPDATETRIGGER

HRTIM Private Macros

IS_HRTIM_TIMERINDEX

IS_HRTIM_TIMING_UNIT

IS_HRTIM_TIMERID

IS_HRTIM_COMPAREUNIT

IS_HRTIM_CAPTUREUNIT

IS_HRTIM_OUTPUT

IS_HRTIM_TIMER_OUTPUT

IS_HRTIM_EVENT

IS_HRTIM_FAULT

IS_HRTIM_PRESCALERRATIO

IS_HRTIM_MODE

IS_HRTIM_MODE_ONEPULSE

IS_HRTIM_HALFMODE

IS_HRTIM_SYNCSTART

IS_HRTIM_SYNCRESET

IS_HHRTIM_DACSYNC

IS_HRTIM_PRELOAD

IS_HRTIM_UPDATEGATING_MASTER

IS_HRTIM_UPDATEGATING_TIM

IS_HRTIM_TIMERBURSTMODE

IS_HRTIM_UPDATEONREPETITION

IS_HRTIM_TIMPUSHPULLMODE

IS_HRTIM_TIMFAULTENABLE

IS_HRTIM_TIMFAULTLOCK

IS_HRTIM_TIMDEADTIMEINSERTION

IS_HRTIM_TIMDELAYEDPROTECTION

IS_HRTIM_TIMUPDATETRIGGER

IS_HRTIM_TIMRESETTRIGGER

IS_HRTIM_TIMUPDATEONRESET

IS_HRTIM_AUTODELAYEDMODE

IS_HRTIM_COMPAREUNIT_AUTODELAYEDMODE

IS_HRTIM_OUTPUTPOLARITY

IS_HRTIM_OUTPUTSET

IS_HRTIM_OUTPUTRESET

IS_HRTIM_OUTPUTIDLEMODE

IS_HRTIM_OUTPUTIDLELEVEL

IS_HRTIM_OUTPUTFAULTLEVEL

IS_HRTIM_OUTPUTCHOPPERMODE

IS_HRTIM_OUTPUTBURSTMODEENTRY

IS_HRTIM_TIMER_CAPTURETRIGGER

IS_HRTIM_TIMEVENTFILTER

IS_HRTIM_TIMEVENTLATCH

IS_HRTIM_TIMDEADTIME_PRESCALERRATIO

IS_HRTIM_TIMDEADTIME_RISINGSIGN

IS_HRTIM_TIMDEADTIME_RISINGLOCK

IS_HRTIM_TIMDEADTIME_RISINGSIGNLOCK

IS_HRTIM_TIMDEADTIME_FALLINGSIGN

IS_HRTIM_TIMDEADTIME_FALLINGLOCK

IS_HRTIM_TIMDEADTIME_FALLINGSIGNLOCK

IS_HRTIM_CHOPPER_PRESCALERRATIO

IS_HRTIM_CHOPPER_DUTYCYCLE

IS_HRTIM_CHOPPER_PULSEWIDTH

IS_HRTIM_SYNCINPUTSOURCE

IS_HRTIM_SYNCOUTPUTSOURCE

IS_HRTIM_SYNCOUTPUTPOLARITY

IS_HRTIM_EVENTSRC

IS_HRTIM_EVENTPOLARITY

IS_HRTIM_EVENTSENSITIVITY

IS_HRTIM_EVENTFASTMODE

IS_HRTIM_EVENTFILTER

IS_HRTIM_EVENTPRESCALER

IS_HRTIM_FAULTSOURCE

IS_HRTIM_FAULTPOLARITY

IS_HRTIM_FAULTFILTER

IS_HRTIM_FAULTLOCK

IS_HRTIM_FAULTPRESCALER

IS_HRTIM_BURSTMODE

IS_HRTIM_BURSTMODECLOCKSOURCE

IS_HRTIM_HRTIM_BURSTMODEPRESCALER

IS_HRTIM_BURSTMODEPRELOAD

IS_HRTIM_BURSTMODETRIGGER

IS_HRTIM_ADCTRIGGERUPDATE

IS_HRTIM_CALIBRATIONRATE

IS_HRTIM_TIMER_BURSTDMA

IS_HRTIM_BURSTMODECTL

IS_HRTIM_TIMERUPDATE

IS_HRTIM_TIMERRESET

IS_HRTIM_IT

IS_HRTIM_MASTER_IT

IS_HRTIM_TIM_IT

IS_HRTIM_MASTER_DMA

IS_HRTIM_TIM_DMA

HRTIM Register Preload Enable

HRTIM_PRELOAD_DISABLED	Preload disabled: the write access is directly done into the active register
HRTIM_PRELOAD_ENABLED	Preload enabled: the write access is done into the preload register

HRTIM Reset On Sync Input Event

HRTIM_SYNCRESET_DISABLED	Synchronization input event has effect on the timer
HRTIM_SYNCRESET_ENABLED	Synchronization input event resets the timer

HRTIM Simple OC Mode

HRTIM_BASICOCMODE_TOGGLE	Output toggles when the timer counter reaches the compare value
HRTIM_BASICOCMODE_INACTIVE	Output forced to active level when the timer counter reaches the compare value
HRTIM_BASICOCMODE_ACTIVE	Output forced to inactive level when the timer counter reaches the compare value

IS_HRTIM_BASICOCMODE

HRTIM Software Timer Reset

HRTIM_TIMERRESET_MASTER	Resets the master timer counter
HRTIM_TIMERRESET_TIMER_A	Resets the timer A counter
HRTIM_TIMERRESET_TIMER_B	Resets the timer B counter
HRTIM_TIMERRESET_TIMER_C	Resets the timer C counter
HRTIM_TIMERRESET_TIMER_D	Resets the timer D counter
HRTIM_TIMERRESET_TIMER_E	Resets the timer E counter

HRTIM Software Timer Update

HRTIM_TIMERUPDATE_MASTER	Forces an immediate transfer from the preload to the active register in the master timer
HRTIM_TIMERUPDATE_A	Forces an immediate transfer from the preload to the active register in the timer A
HRTIM_TIMERUPDATE_B	Forces an immediate transfer from the preload to the active register in the timer B
HRTIM_TIMERUPDATE_C	Forces an immediate transfer from the preload to the active register in the timer C
HRTIM_TIMERUPDATE_D	Forces an immediate transfer from the preload to the active register in the timer D
HRTIM_TIMERUPDATE_E	Forces an immediate transfer from the preload to the active register in the timer E

HRTIM Start On Sync Input Event

HRTIM_SYNCSTART_DISABLED	Synchronization input event has effect on the timer
HRTIM_SYNCSTART_ENABLED	Synchronization input event starts the timer

HRTIM Synchronization Input Source

HRTIM_SYNCINPUTSOURCE_NONE	disabled. HRTIM is not synchronized and runs in standalone mode
HRTIM_SYNCINPUTSOURCE_INTERNALEVENT	The HRTIM is synchronized with the on-chip timer
HRTIM_SYNCINPUTSOURCE_EXTERNALEVENT	A positive pulse on SYNCIN input triggers the HRTIM

HRTIM Synchronization Options

HRTIM_SYNCOPTION_NONE	HRTIM instance doesn't handle external synchronization signals (SYNCIN, SYNCOUT)
HRTIM_SYNCOPTION_MASTER	HRTIM instance acts as a MASTER, i.e. generates external synchronization output (SYNCOUT)
HRTIM_SYNCOPTION_SLAVE	HRTIM instance acts as a SLAVE, i.e. it is synchronized by external sources (SYNCIN)

HRTIM Synchronization Output Polarity

HRTIM_SYNCOUTPUTPOLARITY_NONE	Synchronization output event is disabled
HRTIM_SYNCOUTPUTPOLARITY_POSITIVE	SCOUT pin has a low idle level and issues a positive pulse of 16 fHRTIM

	clock cycles length for the synchronization
HRTIM_SYNCOUTPUTPOLARITY_NEGATIVE	SCOUT pin has a high idle level and issues a negative pulse of 16 fHRTIM clock cycles length for the synchronization

HRTIM Synchronization Output Source

HRTIM_SYNCOUTPUTSOURCE_MASTER_START	A pulse is sent on the SYNCOUT output upon master timer start event
HRTIM_SYNCOUTPUTSOURCE_MASTER_CMP1	A pulse is sent on the SYNCOUT output upon master timer compare 1 event
HRTIM_SYNCOUTPUTSOURCE_TIMA_START	A pulse is sent on the SYNCOUT output upon timer A start or reset events
HRTIM_SYNCOUTPUTSOURCE_TIMA_CMP1	A pulse is sent on the SYNCOUT output upon timer A compare 1 event

HRTIM Timer Burst Mode

HRTIM_TIMERBURSTMODE_MAINTAINCLOCK	Timer counter clock is maintained and the timer operates normally
HRTIM_TIMERBURSTMODE_RESETCOUNTER	Timer counter clock is stopped and the counter is reset

HRTIM Timer Deadtime Insertion

HRTIM_TIMDEADTIMEINSERTION_DISABLED	Output 1 and output 2 signals are independent
HRTIM_TIMDEADTIMEINSERTION_ENABLED	Deadtime is inserted between output 1 and output 2

HRTIM Timer Delayed Protection Mode

HRTIM_TIMDELAYEDPROTECTION_DISABLED	No action
HRTIM_TIMDELAYEDPROTECTION_DELAYEDOUT1_EEV68	Output 1 delayed Idle on external Event 6 or 8
HRTIM_TIMDELAYEDPROTECTION_DELAYEDOUT2_EEV68	Output 2 delayed Idle on external Event 6 or 8
HRTIM_TIMDELAYEDPROTECTION_DELAYEDBOTH_EEV68	Output 1 and output 2 delayed Idle on external Event 6 or 8
HRTIM_TIMDELAYEDPROTECTION_BALANCED_EEV68	Balanced Idle on external Event 6 or 8
HRTIM_TIMDELAYEDPROTECTION_DELAYEDOUT1_DEEV79	Output 1 delayed Idle on external Event 7 or 9

HRTIM_TIMDELAYEDPROTECTION_DELAYEDOUT2_DEEV79	Output 2 delayed Idle on external Event 7 or 9
HRTIM_TIMDELAYEDPROTECTION_DELAYEDBOTH_EEV79	Output 1 and output2 delayed Idle on external Event 7 or 9
HRTIM_TIMDELAYEDPROTECTION_BALANCED_EEV79	Balanced Idle on external Event 7 or 9

HRTIM Timer External Event Filter

HRTIM_TIMEVENTFILTER_NONE	
HRTIM_TIMEVENTFILTER_BLANKINGCMP1	Blanking from counter reset/roll-over to Compare 1
HRTIM_TIMEVENTFILTER_BLANKINGCMP2	Blanking from counter reset/roll-over to Compare 2
HRTIM_TIMEVENTFILTER_BLANKINGCMP3	Blanking from counter reset/roll-over to Compare 3
HRTIM_TIMEVENTFILTER_BLANKINGCMP4	Blanking from counter reset/roll-over to Compare 4
HRTIM_TIMEVENTFILTER_BLANKINGFLTR1	Blanking from another timing unit: TIMFLTR1 source
HRTIM_TIMEVENTFILTER_BLANKINGFLTR2	Blanking from another timing unit: TIMFLTR2 source
HRTIM_TIMEVENTFILTER_BLANKINGFLTR3	Blanking from another timing unit: TIMFLTR3 source
HRTIM_TIMEVENTFILTER_BLANKINGFLTR4	Blanking from another timing unit: TIMFLTR4 source
HRTIM_TIMEVENTFILTER_BLANKINGFLTR5	Blanking from another timing unit: TIMFLTR5 source
HRTIM_TIMEVENTFILTER_BLANKINGFLTR6	Blanking from another timing unit: TIMFLTR6 source
HRTIM_TIMEVENTFILTER_BLANKINGFLTR7	Blanking from another timing unit: TIMFLTR7 source
HRTIM_TIMEVENTFILTER_BLANKINGFLTR8	Blanking from another timing unit: TIMFLTR8 source
HRTIM_TIMEVENTFILTER_WINDOWINGCMP2	Windowing from counter reset/roll-over to Compare 2
HRTIM_TIMEVENTFILTER_WINDOWINGCMP3	Windowing from counter reset/roll-over to Compare 3
HRTIM_TIMEVENTFILTER_WINDOWINGTIM	Windowing from another timing unit: TIMWIN source

HRTIM Timer External Event Latch

HRTIM_TIMEVENTLATCH_DISABLED	Event is ignored if it happens during a blank, or passed through during a window
HRTIM_TIMEVENTLATCH_ENABLED	Event is latched and delayed till the end of the

blanking or windowing period

HRTIM Timer Fault Enabling

HRTIM_TIMFAULTENABLE_NONE	No fault enabled
HRTIM_TIMFAULTENABLE_FAULT1	Fault 1 enabled
HRTIM_TIMFAULTENABLE_FAULT2	Fault 2 enabled
HRTIM_TIMFAULTENABLE_FAULT3	Fault 3 enabled
HRTIM_TIMFAULTENABLE_FAULT4	Fault 4 enabled
HRTIM_TIMFAULTENABLE_FAULT5	Fault 5 enabled

HRTIM Timer Fault Lock

HRTIM_TIMFAULTLOCK_READWRITE	Timer fault enabling bits are read/write
HRTIM_TIMFAULTLOCK_READONLY	Timer fault enabling bits are read only

HRTIM Timer identifier

HRTIM_TIMERID_MASTER	Master identifier
HRTIM_TIMERID_TIMER_A	Timer A identifier
HRTIM_TIMERID_TIMER_B	Timer B identifier
HRTIM_TIMERID_TIMER_C	Timer C identifier
HRTIM_TIMERID_TIMER_D	Timer D identifier
HRTIM_TIMERID_TIMER_E	Timer E identifier

HRTIM Timer Index

HRTIM_TIMERINDEX_TIMER_A	Index used to access timer A registers
HRTIM_TIMERINDEX_TIMER_B	Index used to access timer B registers
HRTIM_TIMERINDEX_TIMER_C	Index used to access timer C registers
HRTIM_TIMERINDEX_TIMER_D	Index used to access timer D registers
HRTIM_TIMERINDEX_TIMER_E	Index used to access timer E registers
HRTIM_TIMERINDEX_MASTER	Index used to access master registers
HRTIM_TIMERINDEX_COMMON	Index used to access HRTIM common registers

HRTIM Timer Output

HRTIM_OUTPUT_TA1	Timer A - Output 1 identifier
HRTIM_OUTPUT_TA2	Timer A - Output 2 identifier
HRTIM_OUTPUT_TB1	Timer B - Output 1 identifier
HRTIM_OUTPUT_TB2	Timer B - Output 2 identifier
HRTIM_OUTPUT_TC1	Timer C - Output 1 identifier
HRTIM_OUTPUT_TC2	Timer C - Output 2 identifier
HRTIM_OUTPUT_TD1	Timer D - Output 1 identifier
HRTIM_OUTPUT_TD2	Timer D - Output 2 identifier
HRTIM_OUTPUT_TE1	Timer E - Output 1 identifier

HRTIM_OUTPUT_TE2	Timer E - Output 2 identifier

HRTIM Timer Push Pull Mode

HRTIM_TIMPUSHPULLMODE_DISABLED	Push-Pull mode disabled
HRTIM_TIMPUSHPULLMODE_ENABLED	Push-Pull mode enabled

HRTIM Timer Repetition Update

HRTIM_UPDATEONREPETITION_DISABLED	Update on repetition disabled
HRTIM_UPDATEONREPETITION_ENABLED	Update on repetition enabled

HRTIM Timer Reset Trigger

HRTIM_TIMRESETTRIGGER_NONE	No counter reset trigger
HRTIM_TIMRESETTRIGGER_UPDATE	The timer counter is reset upon update event
HRTIM_TIMRESETTRIGGER_CMP2	The timer counter is reset upon Timer Compare 2 event
HRTIM_TIMRESETTRIGGER_CMP4	The timer counter is reset upon Timer Compare 4 event
HRTIM_TIMRESETTRIGGER_MASTER_PER	The timer counter is reset upon master timer period event
HRTIM_TIMRESETTRIGGER_MASTER_CMP1	The timer counter is reset upon master timer Compare 1 event
HRTIM_TIMRESETTRIGGER_MASTER_CMP2	The timer counter is reset upon master timer Compare 2 event
HRTIM_TIMRESETTRIGGER_MASTER_CMP3	The timer counter is reset upon master timer Compare 3 event
HRTIM_TIMRESETTRIGGER_MASTER_CMP4	The timer counter is reset upon master timer Compare 4 event
HRTIM_TIMRESETTRIGGER_EEV_1	The timer counter is reset upon external event 1
HRTIM_TIMRESETTRIGGER_EEV_2	The timer counter is reset upon external event 2
HRTIM_TIMRESETTRIGGER_EEV_3	The timer counter is reset upon external event 3
HRTIM_TIMRESETTRIGGER_EEV_4	The timer counter is reset upon external event 4
HRTIM_TIMRESETTRIGGER_EEV_5	The timer counter is reset upon external event 5
HRTIM_TIMRESETTRIGGER_EEV_6	The timer counter is reset upon external event 6
HRTIM_TIMRESETTRIGGER_EEV_7	The timer counter is reset upon external event 7
HRTIM_TIMRESETTRIGGER_EEV_8	The timer counter is reset upon external event 8
HRTIM_TIMRESETTRIGGER_EEV_9	The timer counter is reset upon external

	event 9
HRTIM_TIMRESETTRIGGER_EEV_10	The timer counter is reset upon external event 10
HRTIM_TIMRESETTRIGGER_OTHER1_CMP1	The timer counter is reset upon other timer Compare 1 event
HRTIM_TIMRESETTRIGGER_OTHER1_CMP2	The timer counter is reset upon other timer Compare 2 event
HRTIM_TIMRESETTRIGGER_OTHER1_CMP4	The timer counter is reset upon other timer Compare 4 event
HRTIM_TIMRESETTRIGGER_OTHER2_CMP1	The timer counter is reset upon other timer Compare 1 event
HRTIM_TIMRESETTRIGGER_OTHER2_CMP2	The timer counter is reset upon other timer Compare 2 event
HRTIM_TIMRESETTRIGGER_OTHER2_CMP4	The timer counter is reset upon other timer Compare 4 event
HRTIM_TIMRESETTRIGGER_OTHER3_CMP1	The timer counter is reset upon other timer Compare 1 event
HRTIM_TIMRESETTRIGGER_OTHER3_CMP2	The timer counter is reset upon other timer Compare 2 event
HRTIM_TIMRESETTRIGGER_OTHER3_CMP4	The timer counter is reset upon other timer Compare 4 event
HRTIM_TIMRESETTRIGGER_OTHER4_CMP1	The timer counter is reset upon other timer Compare 1 event
HRTIM_TIMRESETTRIGGER_OTHER4_CMP2	The timer counter is reset upon other timer Compare 2 event
HRTIM_TIMRESETTRIGGER_OTHER4_CMP4	The timer counter is reset upon other timer Compare 4 event

HRTIM Timer Reset Update

HRTIM_TIMUPDATEONRESET_DISABLED	Update by timer x reset / roll-over disabled
HRTIM_TIMUPDATEONRESET_ENABLED	Update by timer x reset / roll-over enabled

HRTIM Timer Update Trigger

HRTIM_TIMUPDATETRIGGER_NONE	Register update is disabled
HRTIM_TIMUPDATETRIGGER_MASTER	Register update is triggered by the master timer update
HRTIM_TIMUPDATETRIGGER_TIMER_A	Register update is triggered by the timer A update
HRTIM_TIMUPDATETRIGGER_TIMER_B	Register update is triggered by the timer B update
HRTIM_TIMUPDATETRIGGER_TIMER_C	Register update is triggered by the timer C update
HRTIM_TIMUPDATETRIGGER_TIMER_D	Register update is triggered by the timer D update
HRTIM_TIMUPDATETRIGGER_TIMER_E	Register update is triggered by the timer E

update

HRTIM Timing Unit DMA Request Enable

HRTIM_TIM_DMA_NONE	No DMA request enable
HRTIM_TIM_DMA_CMP1	Timer compare 1 DMA request enable
HRTIM_TIM_DMA_CMP2	Timer compare 2 DMA request enable
HRTIM_TIM_DMA_CMP3	Timer compare 3 DMA request enable
HRTIM_TIM_DMA_CMP4	Timer compare 4 DMA request enable
HRTIM_TIM_DMA_REP	Timer repetition DMA request enable
HRTIM_TIM_DMA_UPD	Timer update DMA request enable
HRTIM_TIM_DMA_CPT1	Timer capture 1 DMA request enable
HRTIM_TIM_DMA_CPT2	Timer capture 2 DMA request enable
HRTIM_TIM_DMA_SET1	Timer output 1 set DMA request enable
HRTIM_TIM_DMA_RST1	Timer output 1 reset DMA request enable
HRTIM_TIM_DMA_SET2	Timer output 2 set DMA request enable
HRTIM_TIM_DMA_RST2	Timer output 2 reset DMA request enable
HRTIM_TIM_DMA_RST	Timer reset DMA request enable
HRTIM_TIM_DMA_DLYPRT	Timer delay protection DMA request enable

HRTIM Timing Unit Interrupt Enable

HRTIM_TIM_IT_NONE	No interrupt enabled
HRTIM_TIM_IT_CMP1	Timer compare 1 interrupt enable
HRTIM_TIM_IT_CMP2	Timer compare 2 interrupt enable
HRTIM_TIM_IT_CMP3	Timer compare 3 interrupt enable
HRTIM_TIM_IT_CMP4	Timer compare 4 interrupt enable
HRTIM_TIM_IT_REP	Timer repetition interrupt enable
HRTIM_TIM_IT_UPD	Timer update interrupt enable
HRTIM_TIM_IT_CPT1	Timer capture 1 interrupt enable
HRTIM_TIM_IT_CPT2	Timer capture 2 interrupt enable
HRTIM_TIM_IT_SET1	Timer output 1 set interrupt enable
HRTIM_TIM_IT_RST1	Timer output 1 reset interrupt enable
HRTIM_TIM_IT_SET2	Timer output 2 set interrupt enable
HRTIM_TIM_IT_RST2	Timer output 2 reset interrupt enable
HRTIM_TIM_IT_RST	Timer reset interrupt enable
HRTIM_TIM_IT_DLYPRT	Timer delay protection interrupt enable

HRTIM Timing Unit Interrupt Flag

HRTIM_TIM_FLAG_CMP1	Timer compare 1 interrupt flag
HRTIM_TIM_FLAG_CMP2	Timer compare 2 interrupt flag

HRTIM_TIM_FLAG_CMP3	Timer compare 3 interrupt flag
HRTIM_TIM_FLAG_CMP4	Timer compare 4 interrupt flag
HRTIM_TIM_FLAG_REP	Timer repetition interrupt flag
HRTIM_TIM_FLAG_UPD	Timer update interrupt flag
HRTIM_TIM_FLAG_CPT1	Timer capture 1 interrupt flag
HRTIM_TIM_FLAG_CPT2	Timer capture 2 interrupt flag
HRTIM_TIM_FLAG_SET1	Timer output 1 set interrupt flag
HRTIM_TIM_FLAG_RST1	Timer output 1 reset interrupt flag
HRTIM_TIM_FLAG_SET2	Timer output 2 set interrupt flag
HRTIM_TIM_FLAG_RST2	Timer output 2 reset interrupt flag
HRTIM_TIM_FLAG_RST	Timer reset interrupt flag
HRTIM_TIM_FLAG_DLYPRT	Timer delay protection interrupt flag

HRTIM Update Gating

HRTIM_UPDATEGATING_INDEPENDENT	Update done independently from the DMA burst transfer completion
HRTIM_UPDATEGATING_DMABURST	Update done when the DMA burst transfer is completed
HRTIM_UPDATEGATING_DMABURST_UPDATE	Update done on timer roll-over following a DMA burst transfer completion
HRTIM_UPDATEGATING_UPDEN1	Slave timer only - Update done on a rising edge of HRTIM update enable input 1
HRTIM_UPDATEGATING_UPDEN2	Slave timer only - Update done on a rising edge of HRTIM update enable input 2
HRTIM_UPDATEGATING_UPDEN3	Slave timer only - Update done on a rising edge of HRTIM update enable input 3
HRTIM_UPDATEGATING_UPDEN1_UPDATE	Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 1
HRTIM_UPDATEGATING_UPDEN2_UPDATE	Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 2
HRTIM_UPDATEGATING_UPDEN3_UPDATE	Slave timer only - Update done on the update event following a rising edge of HRTIM update enable input 3

22 HAL I2C Generic Driver

22.1 I2C Firmware driver registers structures

22.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32f3xx_hal_i2c.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::Timing***
  Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual.
- ***uint32_t I2C_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t I2C_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C_addressing_mode***.
- ***uint32_t I2C_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C_dual_addressing_mode***.
- ***uint32_t I2C_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- ***uint32_t I2C_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of ***I2C_own_address2_masks***.
- ***uint32_t I2C_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***I2C_general_call_addressing_mode***.
- ***uint32_t I2C_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_nostretch_mode***.

22.1.2 I2C_HandleTypeDef

I2C_HandleTypeDef is defined in the stm32f3xx_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 HAL_I2C_ErrorTypeDef 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 HAL_I2C_ErrorTypeDef I2C_HandleTypeDef::ErrorCode***
  I2C Error code.

22.2 I2C Firmware driver API description

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

22.2.1 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:
   - Enable the I2Cx interface clock
   - I2C pins configuration
     - Enable the clock for the I2C GPIOs

    - Configure I2C pins as alternate function open-drain
  - NVIC configuration if you need to use interrupt process
    - Configure the I2Cx interrupt priority
    - Enable the NVIC I2C IRQ Channel
  - DMA Configuration if you need to use DMA process
    - Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
    - Enable the DMAx interface clock using
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx channel
    - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Configure the Communication Clock Timing, Own Address1, Master Adressing Mode, Dual Addressing Mode, Own Address2, Own Address2 Mask, General call and Nostretch Mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL_I2C_Init() API:
  - These APIs configures also the low level Hardware (GPIO, CLOCK, CORTEX...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 modes of operations are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in no-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 no-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 no-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 no-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

- Write an amount of data in no-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in no-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()

DMA mode IO operation

- Transmit in master mode an amount of data in no-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 no-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 no-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 no-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 Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in no-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback()

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- __HAL_I2C_ENABLE: Enable the I2C peripheral.
- __HAL_I2C_DISABLE: Disable the I2C peripheral.
- __HAL_I2C_GET_FLAG: Checks whether the specified I2C flag is set or not.
- __HAL_I2C_CLEAR_FLAG: Clears the specified I2C pending flag.
- __HAL_I2C_ENABLE_IT: Enables the specified I2C interrupt.
- __HAL_I2C_DISABLE_IT: Disables the specified I2C interrupt.

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

22.2.2 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:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode

  - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx periperal.
- ***HAL_I2C_Init()***
- ***HAL_I2C_DeInit()***
- ***HAL_I2C_MspInit()***
- ***HAL_I2C_MspDeInit()***

22.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There is two mode of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer 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 No_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()*
- *HAL_I2C_EV_IRQHandler()*
- *HAL_I2C_ER_IRQHandler()*
- *HAL_I2C_MasterTxCpltCallback()*
- *HAL_I2C_MasterRxCpltCallback()*
- *HAL_I2C_SlaveTxCpltCallback()*
- *HAL_I2C_SlaveRxCpltCallback()*
- *HAL_I2C_MemTxCpltCallback()*
- *HAL_I2C_MemRxCpltCallback()*
- *HAL_I2C_ErrorCallback()*

22.2.4 Peripheral State and Errors functions

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

- *HAL_I2C_GetState()*
- *HAL_I2C_GetError()*

22.2.5 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

22.2.6 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

22.2.7 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

22.2.8 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

22.2.9 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

22.2.10 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

22.2.11 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

22.2.12 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

22.2.13 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

22.2.14 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

22.2.15 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

22.2.16 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

22.2.17 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

22.2.18 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

22.2.19 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

22.2.20 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

22.2.21 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

22.2.22 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

22.2.23 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

22.2.24 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

22.2.25 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

22.2.26 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

22.2.27 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

22.2.28 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

22.2.29 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 the specified I2C.
Return values	• None

22.2.30 HAL_I2C_MasterTxCpltCallback

Function Name	**void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Master Tx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

22.2.31 HAL_I2C_MasterRxCpltCallback

Function Name	**void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Master Rx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

22.2.32 HAL_I2C_SlaveTxCpltCallback

Function Name	**void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Slave Tx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

22.2.33 HAL_I2C_SlaveRxCpltCallback

Function Name	**void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Slave Rx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

22.2.34 HAL_I2C_MemTxCpltCallback

Function Name	**void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Memory Tx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

22.2.35 HAL_I2C_MemRxCpltCallback

Function Name	**void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)**
Function Description	Memory Rx Transfer completed callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

22.2.36 HAL_I2C_ErrorCallback

Function Name	**void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)**
Function Description	I2C error callback.
Parameters	• **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	• None

22.2.37 HAL_I2C_GetState

Function Name	**HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)**
Function Description	Returns the I2C state.
Parameters	• **hi2c:** I2C handle
Return values	• HAL state

22.2.38 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

22.3 I2C Firmware driver defines

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

22.3.1 I2C

I2C

I2C addressing mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

IS_I2C_ADDRESSING_MODE

I2C dual addressing mode

I2C_DUALADDRESS_DISABLED

I2C_DUALADDRESS_ENABLED

IS_I2C_DUAL_ADDRESS

I2C Exported Macros

__HAL_I2C_RESET_HANDLE_STATE

Description:

- Reset I2C handle state.

Parameters:

- __HANDLE__: I2C handle.

Return value:

- None:

__HAL_I2C_ENABLE_IT

Description:

- Enable the specified I2C interrupts.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2Cx where x: 1, 2, or 3 to select the I2C peripheral.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  – I2C_IT_ERRI: Errors interrupt enable
  – I2C_IT_TCI: Transfer complete interrupt enable
  – I2C_IT_STOPI: STOP detection interrupt enable
  – I2C_IT_NACKI: NACK received interrupt enable
  – I2C_IT_ADDRI: Address match interrupt enable
  – I2C_IT_RXI: RX interrupt enable

- I2C_IT_TXI: TX interrupt enable

Return value:

- None:

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupts.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2Cx where x: 1, 2, or 3 to select the I2C peripheral.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - I2C_IT_ERRI: Errors interrupt enable
  - I2C_IT_TCI: Transfer complete interrupt enable
  - I2C_IT_STOPI: STOP detection interrupt enable
  - I2C_IT_NACKI: NACK received interrupt enable
  - I2C_IT_ADDRI: Address match interrupt enable
  - I2C_IT_RXI: RX interrupt enable
  - I2C_IT_TXI: TX interrupt enable

Return value:

- None:

__HAL_I2C_GET_IT_SOURCE

Description:

- Check if the specified I2C interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2Cx where x: 1, 2, or 3 to select the I2C peripheral.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C_IT_ERRI: Errors interrupt enable
  - I2C_IT_TCI: Transfer complete interrupt enable
  - I2C_IT_STOPI: STOP detection interrupt enable
  - I2C_IT_NACKI: NACK received interrupt enable
  - I2C_IT_ADDRI: Address match interrupt enable
  - I2C_IT_RXI: RX interrupt enable
  - I2C_IT_TXI: TX interrupt enable

Return value:

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

__HAL_I2C_GET_FLAG

Description:

- Check whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2Cx where x: 1, 2, or 3 to select the I2C peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2C_FLAG_TXE: Transmit data register empty
  - I2C_FLAG_TXIS: Transmit interrupt status
  - I2C_FLAG_RXNE: Receive data register not empty
  - I2C_FLAG_ADDR: Address matched (slave mode)
  - I2C_FLAG_AF: Acknowledge failure received flag
  - I2C_FLAG_STOPF: STOP detection flag
  - I2C_FLAG_TC: Transfer complete (master mode)
  - I2C_FLAG_TCR: Transfer complete reload
  - I2C_FLAG_BERR: Bus error
  - I2C_FLAG_ARLO: Arbitration lost
  - I2C_FLAG_OVR: Overrun/Underrun
  - I2C_FLAG_PECERR: PEC error in reception
  - I2C_FLAG_TIMEOUT: Timeout or Tlow detection flag
  - I2C_FLAG_ALERT: SMBus alert
  - I2C_FLAG_BUSY: Bus busy
  - I2C_FLAG_DIR: Transfer direction (slave mode)

Return value:

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

__HAL_I2C_CLEAR_FLAG

Description:

- Clears the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle. This parameter can be I2Cx where x: 1, 2, or 3 to select the I2C peripheral.

- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_ADDR: Address matched (slave mode)
  - I2C_FLAG_AF: Acknowledge failure flag
  - I2C_FLAG_STOPF: STOP detection flag
  - I2C_FLAG_BERR: Bus error
  - I2C_FLAG_ARLO: Arbitration lost
  - I2C_FLAG_OVR: Overrun/Underrun
  - I2C_FLAG_PECERR: PEC error in reception
  - I2C_FLAG_TIMEOUT: Timeout or Tlow detection flag
  - I2C_FLAG_ALERT: SMBus alert

Return value:

- None:

__HAL_I2C_ENABLE

Description:

- Enable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None:

__HAL_I2C_DISABLE

Description:

- Disable the specified I2C peripheral.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None:

__HAL_I2C_RESET_CR2

__HAL_I2C_MEM_ADD_MSB

__HAL_I2C_MEM_ADD_LSB

__HAL_I2C_GENERATE_START

IS_I2C_OWN_ADDRESS1

IS_I2C_OWN_ADDRESS2

I2C Flag definition

I2C_FLAG_TXE

I2C_FLAG_TXIS

I2C_FLAG_RXNE

I2C_FLAG_ADDR

I2C_FLAG_AF

I2C_FLAG_STOPF

I2C_FLAG_TC

I2C_FLAG_TCR

I2C_FLAG_BERR

I2C_FLAG_ARLO

I2C_FLAG_OVR

I2C_FLAG_PECERR

I2C_FLAG_TIMEOUT

I2C_FLAG_ALERT

I2C_FLAG_BUSY

I2C_FLAG_DIR

I2C general call addressing mode

I2C_GENERALCALL_DISABLED

I2C_GENERALCALL_ENABLED

IS_I2C_GENERAL_CALL

I2C Interrupt configuration definition

I2C_IT_ERRI

I2C_IT_TCI

I2C_IT_STOPI

I2C_IT_NACKI

I2C_IT_ADDRI

I2C_IT_RXI

I2C_IT_TXI

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

IS_I2C_MEMADD_SIZE

I2C nostretch mode

I2C_NOSTRETCH_DISABLED

I2C_NOSTRETCH_ENABLED

IS_I2C_NO_STRETCH

I2C own address2 masks

I2C_OA2_NOMASK

I2C_OA2_MASK01

I2C_OA2_MASK02

I2C_OA2_MASK03

I2C_OA2_MASK04

I2C_OA2_MASK05

I2C_OA2_MASK06

I2C_OA2_MASK07

IS_I2C_OWN_ADDRESS2_MASK

I2C Private Define

TIMING_CLEAR_MASK

I2C_TIMEOUT_ADDR

I2C_TIMEOUT_BUSY

I2C_TIMEOUT_DIR

I2C_TIMEOUT_RXNE

I2C_TIMEOUT_STOPF

I2C_TIMEOUT_TC

I2C_TIMEOUT_TCR

I2C_TIMEOUT_TXIS

I2C_TIMEOUT_FLAG

I2C ReloadEndMode definition

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

IS_TRANSFER_MODE

I2C StartStopMode definition

I2C_NO_STARTSTOP

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

IS_TRANSFER_REQUEST

23 HAL I2C Extension Driver

23.1 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

23.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32F3XX devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode

23.1.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function HAL_I2CEx_AnalogFilter_Config()
2. Configure I2C Digital noise filter using the function HAL_I2CEx_DigitalFilter_Config()
3. Configure the enabling or disabling of I2C Wake Up Mode using these functions :
   - HAL_I2CEx_EnableWakeUp()
   - HAL_I2CEx_DisableWakeUp()

23.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters
- Configure Wake Up Feature
- ***HAL_I2CEx_AnalogFilter_Config()***
- ***HAL_I2CEx_DigitalFilter_Config()***
- ***HAL_I2CEx_EnableWakeUp()***
- ***HAL_I2CEx_DisableWakeUp()***

23.1.4 HAL_I2CEx_AnalogFilter_Config

Function Name	**HAL_StatusTypeDef HAL_I2CEx_AnalogFilter_Config (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)**
Function Description	Configure I2C Analog noise filter.
Parameters	• **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral. • **AnalogFilter:** new state of the Analog filter.
Return values	• HAL status

23.1.5 HAL_I2CEx_DigitalFilter_Config

Function Name	**HAL_StatusTypeDef HAL_I2CEx_DigitalFilter_Config (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)**
Function Description	Configure I2C Digital noise filter.
Parameters	• **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral. • **DigitalFilter:** Coefficient of digital noise filter between 0x00 and 0x0F.
Return values	• HAL status

23.1.6 HAL_I2CEx_EnableWakeUp

Function Name	**HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)**
Function Description	Enable I2C wakeup from stop mode.
Parameters	• **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	• HAL status

23.1.7 HAL_I2CEx_DisableWakeUp

Function Name	**HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)**
Function Description	Disable I2C wakeup from stop mode.
Parameters	• **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
Return values	• HAL status

23.2 I2CEx Firmware driver defines

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

23.2.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLED

I2C_ANALOGFILTER_DISABLED

IS_I2C_ANALOG_FILTER

I2C Extended Digital Filter

IS_I2C_DIGITAL_FILTER

24 HAL I2S Generic Driver

24.1 I2S Firmware driver registers structures

24.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32f3xx_hal_i2s.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t CPOL***
- ***uint32_t ClockSource***
- ***uint32_t FullDuplexMode***

Field Documentation

- ***uint32_t I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_Mode***
- ***uint32_t I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_Standard***
- ***uint32_t I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_Data_Format***
- ***uint32_t I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_MCLK_Output***
- ***uint32_t I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_Audio_Frequency***
- ***uint32_t I2S_InitTypeDef::CPOL***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_Clock_Polarity***
- ***uint32_t I2S_InitTypeDef::ClockSource***
  Specifies the I2S Clock Source. This parameter can be a value of ***I2S_Clock_Source***
- ***uint32_t I2S_InitTypeDef::FullDuplexMode***
  Specifies the I2S FullDuplex mode. This parameter can be a value of ***I2S_FullDuplex_Mode***

24.1.2 I2S_HandleTypeDef

I2S_HandleTypeDef is defined in the stm32f3xx_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 HAL_I2S_ErrorTypeDef ErrorCode***

Field Documentation

- ***SPI_TypeDef* I2S_HandleTypeDef::Instance***
  I2S registers base address
- ***I2S_InitTypeDef I2S_HandleTypeDef::Init***
  I2S communication parameters
- ***uint16_t* I2S_HandleTypeDef::pTxBuffPtr***
  Pointer to I2S Tx transfer buffer
- ***__IO uint16_t I2S_HandleTypeDef::TxXferSize***
  I2S Tx transfer size
- ***__IO uint16_t I2S_HandleTypeDef::TxXferCount***
  I2S Tx transfer Counter
- ***uint16_t* I2S_HandleTypeDef::pRxBuffPtr***
  Pointer to I2S Rx transfer buffer
- ***__IO uint16_t I2S_HandleTypeDef::RxXferSize***
  I2S Rx transfer size
- ***__IO uint16_t I2S_HandleTypeDef::RxXferCount***
  I2S Rx transfer counter (This field is initialized at the same value as transfer size at the beginning of the transfer and decremented when a sample is received. NbSamplesReceived = RxBufferSize-RxBufferCount)
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx***
  I2S Tx DMA handle parameters
- ***DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx***
  I2S Rx DMA handle parameters
- ***__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock***
  I2S locking object
- ***__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State***
  I2S communication state
- ***__IO HAL_I2S_ErrorTypeDef I2S_HandleTypeDef::ErrorCode***
  I2S Error code

24.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

24.2.1 How to use this driver

The I2S HAL driver can be used as follows:

1. Declare a I2S_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
   a. Enable the SPIx interface clock.
   b. I2S pins configuration:
      – Enable the clock for the I2S GPIOs.
      – Configure these I2S pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_I2S_Transmit_IT() and HAL_I2S_Receive_IT() APIs).
      – Configure the I2Sx interrupt priority.
      – Enable the NVIC I2S IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_I2S_Transmit_DMA() and HAL_I2S_Receive_DMA() APIs:
      – Declare a DMA handle structure for the Tx/Rx channel.
      – Enable the DMAx interface clock.
      – Configure the declared DMA handle structure with the required Tx/Rx parameters.
      – Configure the DMA Tx/Rx Channel.
      – Associate the 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. Make sure that either: I2S clock is configured based on SYSCLK or External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32f3xx_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 I2S HAL driver.

- __HAL_I2S_ENABLE: Enable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_DISABLE: Disable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_ENABLE_IT : Enable the specified I2S interrupts
- __HAL_I2S_DISABLE_IT : Disable the specified I2S interrupts
- __HAL_I2S_GET_FLAG: Check whether the specified I2S flag is set or not

You can refer to the I2S HAL driver header file for more useful macros

24.2.2 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
  - Full duplex mode

- 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()***

24.2.3 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_IRQHandler()***
- ***HAL_I2S_TxHalfCpltCallback()***
- ***HAL_I2S_TxCpltCallback()***
- ***HAL_I2S_RxHalfCpltCallback()***
- ***HAL_I2S_RxCpltCallback()***
- ***HAL_I2S_ErrorCallback()***
- ***HAL_I2S_FullDuplex_IRQHandler()***
- ***HAL_I2S_TxRxCpltCallback()***

24.2.4 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()*
- *HAL_I2S_DMAPause()*
- *HAL_I2S_DMAResume()*
- *HAL_I2S_DMAStop()*

24.2.5 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 • **hi2s:** I2S handle
Return values	• HAL status • HAL status

24.2.6 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

24.2.7 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

24.2.8 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

24.2.9 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).

24.2.10 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.

24.2.11 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).

24.2.12 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.

24.2.13 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).

24.2.14 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).

24.2.15 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	• HAL status

24.2.16 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

24.2.17 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

24.2.18 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

24.2.19 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

24.2.20 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

24.2.21 HAL_I2S_FullDuplex_IRQHandler

Function Name	**void HAL_I2S_FullDuplex_IRQHandler (I2S_HandleTypeDef * hi2s)**
Function Description	This function handles I2S/I2Sext interrupt requests in full-duplex mode.
Parameters	• **hi2s:** I2S handle
Return values	• HAL status

24.2.22 HAL_I2S_TxRxCpltCallback

Function Name	**void HAL_I2S_TxRxCpltCallback (I2S_HandleTypeDef * hi2s)**
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	• **hi2s:** I2S handle
Return values	• None

24.2.23 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

24.2.24 HAL_I2S_GetError

Function Name	**HAL_I2S_ErrorTypeDef 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

24.2.25 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:** : I2S handle
Return values	•	None

24.2.26 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:** : I2S handle
Return values	• None

24.2.27 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:** I2S handle
Return values	• None

24.3 I2S Firmware driver defines

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

24.3.1 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

IS_I2S_AUDIO_FREQ

I2S Clock Polarity

I2S_CPOL_LOW

I2S_CPOL_HIGH

IS_I2S_CPOL

I2S Clock Source

I2S_CLOCK_EXTERNAL

I2S_CLOCK_SYSCLK

IS_I2S_CLOCKSOURCE

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

IS_I2S_DATA_FORMAT

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: I2S handle.

Return value:

- None:

__HAL_I2S_ENABLE

Description:

- Enable or disable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None:

__HAL_I2S_DISABLE

__HAL_I2S_ENABLE_IT

Description:

- Enable or 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_DISABLE_IT

__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_FRE: Frame error 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 Full Duplex Mode

I2S_FULLDUPLEXMODE_DISABLE

I2S_FULLDUPLEXMODE_ENABLE

IS_I2S_FULLDUPLEX_MODE

I2S Interrupt configuration definition

I2S_IT_TXE

I2S_IT_RXNE

I2S_IT_ERR

I2S MCLK Output

I2S_MCLKOUTPUT_ENABLE

I2S_MCLKOUTPUT_DISABLE

IS_I2S_MCLK_OUTPUT

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

IS_I2S_MODE

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

IS_I2S_STANDARD

25 HAL I2S Extension Driver

25.1 I2SEx Firmware driver API description

The following section lists the various functions of the I2SEx library.

25.1.1 I2S Extended features

1. In I2S full duplex mode, each SPI peripheral is able to manage sending and receiving data simultaneously using two data lines. Each SPI peripheral has an extended block called I2Sxext ie. I2S2ext for SPI2 and I2S3ext for SPI3).
2. The Extended block is not a full SPI IP, it is used only as I2S slave to implement full duplex mode. The Extended block uses the same clock sources as its master (refer to the following Figure). +-----------------------+ I2Sx_SCK | | ----------+-->| I2Sx |------------------->I2Sx_SD(in/out) +--|-->| | | | +-----------------------+ | | I2S_WS | | ------>| | | | +-----------------------+ | +-->| | | | I2Sx_ext |------------------->I2Sx_extSD(in/out) +----->| | +-----------------------+
3. Both I2Sx and I2Sx_ext can be configured as transmitters or receivers. Only I2Sx can deliver SCK and WS to I2Sx_ext in full duplex mode, where I2Sx can be I2S2 or I2S3. ==

25.1.2 How to use this driver

Three mode of operations are available within this driver :

Polling mode IO operation

- Send and receive in the same time an amount of data in blocking mode using HAL_I2S_TransmitReceive()

Interrupt mode IO operation

- Send and receive in the same time an amount of data in non blocking mode using HAL_I2S_TransmitReceive_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
- 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 and receive an amount of data in non blocking mode (DMA) using HAL_I2S_TransmitReceive_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
- 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()

25.1.3 Extended features Functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There is two mode of transfer:
   - Blocking mode: The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode: The communication is performed using Interrupts or DMA. These functions return the status of the transfer 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_TransmitReceive()
3. No-Blocking mode functions with Interrupt are:
   - HAL_I2S_TransmitReceive_IT()
   - HAL_I2SFullDuplex_IRQHandler()
4. No-Blocking mode functions with DMA are:
   - HAL_I2S_TransmitReceive_DMA()
5. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
   - HAL_I2S_TxRxCpltCallback()
   - HAL_I2S_TxRxErrorCallback()

- ***HAL_I2SEx_TransmitReceive()***
- ***HAL_I2SEx_TransmitReceive_IT()***
- ***HAL_I2SEx_TransmitReceive_DMA()***

25.1.4 HAL_I2SEx_TransmitReceive

Function Name	**HAL_StatusTypeDef HAL_I2SEx_TransmitReceive**

	(I2S_HandleTypeDef * hi2s, uint16_t * pTxData, uint16_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Full-Duplex Transmit/Receive data in blocking mode.
Parameters	• **hi2s:** I2S handle • **pTxData:** a 16-bit pointer to the Transmit data buffer. • **pRxData:** a 16-bit pointer to the Receive data buffer. • **Size:** number of data sample to be sent: • **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).

25.1.5 HAL_I2SEx_TransmitReceive_IT

Function Name	**HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)**
Function Description	Full-Duplex Transmit/Receive data in non-blocking mode using Interrupt.
Parameters	• **hi2s:** I2S handle • **pTxData:** a 16-bit pointer to the Transmit data buffer. • **pRxData:** a 16-bit pointer to the Receive data buffer. • **Size:** number of data sample to be sent:
Return values	• HAL status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

25.1.6 HAL_I2SEx_TransmitReceive_DMA

Function Name	**HAL_StatusTypeDef HAL_I2SEx_TransmitReceive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)**

Function Description	Full-Duplex Transmit/Receive data in non-blocking mode using DMA.
Parameters	• **hi2s:** I2S handle • **pTxData:** a 16-bit pointer to the Transmit data buffer. • **pRxData:** a 16-bit pointer to the Receive data buffer. • **Size:** number of data sample to be sent:
Return values	• HAL status
Notes	• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

25.2 I2SEx Firmware driver defines

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

25.2.1 I2SEx

I2SEx

I2S Extended Exported Macros

I2SxEXT

__HAL_I2SEXT_ENABLE

Description:

- Enable or disable the specified I2SExt peripheral.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None:

__HAL_I2SEXT_DISABLE

__HAL_I2SEXT_ENABLE_IT

Description:

- Enable or disable the specified I2SExt 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_I2SEXT_DISABLE_IT

__HAL_I2SEXT_GET_IT_SOURCE

Description:

- Checks if the specified I2SExt 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_I2SEXT_GET_FLAG

Description:

- Checks whether the specified I2SExt 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_FRE: Frame error flag
  - I2S_FLAG_CHSIDE: Channel Side flag
  - I2S_FLAG_BSY: Busy flag

Return value:

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

__HAL_I2SEXT_CLEAR_OVRFLAG

Description:

- Clears the I2SExt OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None:

__HAL_I2SEXT_CLEAR_UDRFLAG	**Description:** • Clears the I2SExt UDR pending flag. **Parameters:** • __HANDLE__: specifies the I2S Handle. **Return value:** • None:

26 HAL IRDA Generic Driver

26.1 IRDA Firmware driver registers structures

26.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32f3xx_hal_irda.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint16_t Parity***
- ***uint16_t Mode***
- ***uint8_t Prescaler***
- ***uint16_t PowerMode***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***
  This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hirda->Init.BaudRate)))
- ***uint32_t IRDA_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***IRDAEx_Word_Length***
- ***uint16_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).
- ***uint16_t IRDA_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***IRDA_Mode***
- ***uint8_t IRDA_InitTypeDef::Prescaler***
  Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.
  Note: Prescaler value 0 is forbidden
- ***uint16_t IRDA_InitTypeDef::PowerMode***
  Specifies the IRDA power mode. This parameter can be a value of ***IRDA_Low_Power***

26.1.2 IRDA_HandleTypeDef

IRDA_HandleTypeDef is defined in the stm32f3xx_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***
- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***HAL_IRDA_StateTypeDef State***
- ***HAL_IRDA_ErrorTypeDef 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
- ***uint16_t IRDA_HandleTypeDef::Mask***
  USART RX RDR register mask
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx***
  IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx***
  IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef IRDA_HandleTypeDef::Lock***
  Locking object
- ***HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::State***
  IRDA communication state
- ***HAL_IRDA_ErrorTypeDef IRDA_HandleTypeDef::ErrorCode***
  IRDA Error code

26.2 IRDA Firmware driver API description

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

26.2.1 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 in setting the associated USART or UART in IRDA mode:
   a. Enable the USARTx/UARTx interface clock.
   b. USARTx/UARTx pins configuration:
      – Enable the clock for the USARTx/UARTx GPIOs.
      – Configure these USARTx/UARTx pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
      – Configure the USARTx/UARTx interrupt priority.
      – Enable the NVIC IRDA 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.
3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler in the hirda 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

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

26.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit), the possible IRDA frame formats are as listed in *Table 22: "IRDA frame formats"* .
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

Table 22: 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 |
M1, M0 bits	**PCE bit**	**IRDA frame**

M bit	PCE bit	IRDA frame
10	0	| SB | 7-bit data | STB |
10	1	SB | 6-bit data | PB | STB |

The HAL_IRDA_Init() function follows IRDA configuration procedures (details for the procedures are available in reference manual).

- ***HAL_IRDA_Init()***
- ***HAL_IRDA_DeInit()***
- ***HAL_IRDA_MspInit()***
- ***HAL_IRDA_MspDeInit()***

26.2.3 IO operation functions

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

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
   – Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   – Non Blocking mode: The communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectivelly 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 API's are :
   – HAL_IRDA_Transmit()
   – HAL_IRDA_Receive()
3. Non-Blocking mode API's with Interrupt are :
   – HAL_IRDA_Transmit_IT()
   – HAL_IRDA_Receive_IT()
   – HAL_IRDA_IRQHandler()
   – IRDA_Transmit_IT()
   – IRDA_Receive_IT()
4. Non-Blocking mode functions with DMA are :
   – HAL_IRDA_Transmit_DMA()
   – HAL_IRDA_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
   – HAL_IRDA_TxCpltCallback()
   – 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_IRQHandler()*
- *HAL_IRDA_TxCpltCallback()*
- *HAL_IRDA_RxCpltCallback()*
- *HAL_IRDA_ErrorCallback()*

26.2.4 Peripheral State and Error functions

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

- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IRDA peripheral.
- IRDA_SetConfig() API is used to configure the IRDA communications parameters.
- *HAL_IRDA_GetState()*
- *HAL_IRDA_GetError()*

26.2.5 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 creates the associated handle .
Parameters	• **hirda:** IRDA handle
Return values	• HAL status

26.2.6 HAL_IRDA_DeInit

Function Name	**HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)**
Function Description	DeInitializes the IRDA peripheral.
Parameters	• **hirda:** IRDA handle
Return values	• HAL status

26.2.7 HAL_IRDA_MspInit

Function Name	**void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)**
Function Description	IRDA MSP Init.
Parameters	• **hirda:** IRDA handle
Return values	• None

26.2.8 HAL_IRDA_MspDeInit

Function Name	**void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)**

Function Description	IRDA MSP DeInit.
Parameters	• **hirda:** IRDA handle
Return values	• None

26.2.9 HAL_IRDA_Transmit

Function Name	**HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Send an amount of data in blocking mode.
Parameters	• **hirda:** IRDA handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent • **Timeout:** Duration of the timeout
Return values	• HAL status

26.2.10 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:** IRDA handle • **pData:** pointer to data buffer • **Size:** amount of data to be received • **Timeout:** Duration of the timeout
Return values	• HAL status

26.2.11 HAL_IRDA_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function Description	Send an amount of data in interrupt mode.
Parameters	• **hirda:** IRDA handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

26.2.12 HAL_IRDA_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**

Function Description	Receive an amount of data in interrupt mode.
Parameters	• **hirda:** IRDA handle • **pData:** pointer to data buffer • **Size:** amount of data to be received
Return values	• HAL status

26.2.13 HAL_IRDA_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**
Function Description	Send an amount of data in DMA mode.
Parameters	• **hirda:** IRDA handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

26.2.14 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 DMA mode.
Parameters	• **hirda:** IRDA handle • **pData:** pointer to data buffer • **Size:** amount of data to be received
Return values	• HAL status
Notes	• When the IRDA parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

26.2.15 HAL_IRDA_IRQHandler

Function Name	**void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)**
Function Description	This function handles IRDA interrupt request.
Parameters	• **hirda:** IRDA handle
Return values	• None

26.2.16 HAL_IRDA_TxCpltCallback

Function Name	**void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)**
Function Description	Tx Transfer completed callback.

Parameters	•	**hirda:** irda handle
Return values	•	None

26.2.17 HAL_IRDA_RxCpltCallback

Function Name	**void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)**
Function Description	Rx Transfer completed callback.
Parameters	• **hirda:** irda handle
Return values	• None

26.2.18 HAL_IRDA_ErrorCallback

Function Name	**void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)**
Function Description	IRDA error callback.
Parameters	• **hirda:** IRDA handle
Return values	• None

26.2.19 HAL_IRDA_GetState

Function Name	**HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)**
Function Description	return the IRDA state
Parameters	• **hirda:** irda handle
Return values	• HAL state

26.2.20 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.
Return values	• IRDA Error Code

26.3 IRDA Firmware driver defines

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

26.3.1 IRDA

IRDA

IRDA DMA Rx

IRDA_DMA_RX_DISABLE

IRDA_DMA_RX_ENABLE

IS_IRDA_DMA_RX

IRDA DMA Tx

IRDA_DMA_TX_DISABLE

IRDA_DMA_TX_ENABLE

IS_IRDA_DMA_TX

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE

Description:

- Reset IRDA handle state.

Parameters:

- __HANDLE__: IRDA handle.

Return value:

- None:

__HAL_IRDA_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IRDA_FLAG_REACK: Receive enable ackowledge flag
  - IRDA_FLAG_TEACK: Transmit enable ackowledge flag
  - IRDA_FLAG_BUSY: Busy flag
  - IRDA_FLAG_ABRF: Auto Baud rate detection flag
  - IRDA_FLAG_ABRE: Auto Baud rate detection error flag
  - IRDA_FLAG_TXE: Transmit data register empty flag
  - IRDA_FLAG_TC: Transmission Complete flag
  - IRDA_FLAG_RXNE: Receive data register not empty flag
  - IRDA_FLAG_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_ENABLE_IT

Description:

- Enables the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- __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:

- Disables the specified IRDA interrupt.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- __INTERRUPT__: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA_IT_TXE: Transmit Data Register empty interrupt
  - IRDA_IT_TC: Transmission complete interrupt
  - IRDA_IT_RXNE: Receive Data register not empty interrupt
  - IRDA_IT_IDLE: Idle line detection interrupt
  - IRDA_IT_PE: Parity Error interrupt
  - IRDA_IT_ERR: Error interrupt(Frame

error, noise error, overrun error)

Return value:

- None:

__HAL_IRDA_GET_IT

Description:

- Checks whether the specified IRDA interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- __IT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  – IRDA_IT_TXE: Transmit Data Register empty interrupt
  – IRDA_IT_TC: Transmission complete interrupt
  – IRDA_IT_RXNE: Receive Data register not empty interrupt
  – IRDA_IT_IDLE: Idle line detection interrupt
  – IRDA_IT_ORE: OverRun Error interrupt
  – IRDA_IT_NE: Noise Error interrupt
  – IRDA_IT_FE: Framing Error interrupt
  – IRDA_IT_PE: Parity Error interrupt

Return value:

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

__HAL_IRDA_GET_IT_SOURCE

Description:

- Checks whether the specified IRDA interrupt source is enabled.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- __IT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  – IRDA_IT_TXE: Transmit Data Register empty interrupt
  – IRDA_IT_TC: Transmission complete interrupt
  – IRDA_IT_RXNE: Receive Data register not empty interrupt
  – IRDA_IT_IDLE: Idle line detection interrupt
  – IRDA_IT_ORE: OverRun Error interrupt

- IRDA_IT_NE: Noise Error interrupt
- IRDA_IT_FE: Framing Error interrupt
- IRDA_IT_PE: Parity Error interrupt

Return value:

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

__HAL_IRDA_CLEAR_IT

Description:

- Clears the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - IRDA_CLEAR_PEF: Parity Error Clear Flag
  - IRDA_CLEAR_FEF: Framing Error Clear Flag
  - IRDA_CLEAR_NEF: Noise detected Clear Flag
  - IRDA_CLEAR_OREF: OverRun Error Clear Flag
  - IRDA_CLEAR_TCF: Transmission Complete Clear Flag

Return value:

- None:

__HAL_IRDA_SEND_REQ

Description:

- Set a specific IRDA request flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - IRDA_AUTOBAUD_REQUEST: Auto-Baud Rate Request
  - IRDA_RXDATA_FLUSH_REQUEST: Receive Data flush Request
  - IRDA_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None:

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral

Return value:

- None:

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral

Return value:

- None:

IS_IRDA_BAUDRATE

Description:

- Ensure that IRDA Baud rate is less or equal to maximum value.

Parameters:

- __BAUDRATE__: specifies the IRDA Baudrate set by the user.

Return value:

- True: or False

IS_IRDA_PRESCALER

Description:

- Ensure that IRDA prescaler value is strictly larger than 0.

Parameters:

- __PRESCALER__: specifies the IRDA prescaler value set by the user.

Return value:

- True: or False

IRDA Flags

IRDA_FLAG_REACK

IRDA_FLAG_TEACK

IRDA_FLAG_BUSY

IRDA_FLAG_ABRF

IRDA_FLAG_ABRE

IRDA_FLAG_TXE

IRDA_FLAG_TC

IRDA_FLAG_RXNE

IRDA_FLAG_ORE

IRDA_FLAG_NE

IRDA_FLAG_FE

IRDA_FLAG_PE

IRDA interruptions flag mask

IRDA_IT_MASK

IRDA Interrupts Definition

IRDA_IT_PE

IRDA_IT_TXE

IRDA_IT_TC

IRDA_IT_RXNE

IRDA_IT_IDLE

IRDA_IT_ERR

IRDA_IT_ORE

IRDA_IT_NE

IRDA_IT_FE

IRDA Interruption Clear Flags

IRDA_CLEAR_PEF	Parity Error Clear Flag
IRDA_CLEAR_FEF	Framing Error Clear Flag
IRDA_CLEAR_NEF	Noise detected Clear Flag
IRDA_CLEAR_OREF	OverRun Error Clear Flag
IRDA_CLEAR_TCF	Transmission Complete Clear Flag

IRDA Low Power

IRDA_POWERMODE_NORMAL

IRDA_POWERMODE_LOWPOWER

IS_IRDA_POWERMODE

IRDA Mode

IRDA_MODE_DISABLE

IRDA_MODE_ENABLE

IS_IRDA_MODE

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLED

IRDA_ONE_BIT_SAMPLE_ENABLED

IS_IRDA_ONEBIT_SAMPLE

IRDA Parity

IRDA_PARITY_NONE

IRDA_PARITY_EVEN

IRDA_PARITY_ODD

IS_IRDA_PARITY

IRDA Private Constants

TEACK_REACK_TIMEOUT

IRDA_TXDMA_TIMEOUTVALUE

IRDA_TIMEOUT_VALUE

IRDA_CR1_FIELDS

IRDA Request Parameters

IRDA_AUTOBAUD_REQUEST	Auto-Baud Rate Request
IRDA_RXDATA_FLUSH_REQUEST	Receive Data flush Request
IRDA_TXDATA_FLUSH_REQUEST	Transmit data flush Request
IS_IRDA_REQUEST_PARAMETER	

IRDA State

IRDA_STATE_DISABLE

IRDA_STATE_ENABLE

IS_IRDA_STATE

IRDA Transfer Mode

IRDA_MODE_RX

IRDA_MODE_TX

IRDA_MODE_TX_RX

IS_IRDA_TX_RX_MODE

27 HAL IRDA Extension Driver

27.1 IRDAEx Firmware driver defines

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

27.1.1 IRDAEx

IRDAEx

IRDA Extended Exported Macros

__HAL_IRDA_GETCLOCKSOURCE

Description:

- Reports the IRDA clock source.

Parameters:

- __HANDLE__: specifies the IRDA Handle
- __CLOCKSOURCE__: output variable

Return value:

- IRDA: clocking source, written in __CLOCKSOURCE__.

__HAL_IRDA_MASK_COMPUTATION

Description:

- Computes the mask to apply to retrieve the received data according to the word length and to the parity bits activation.

Parameters:

- __HANDLE__: specifies the IRDA Handle

Return value:

- none:

IRDA Extended Word Length

IRDA_WORDLENGTH_7B

IRDA_WORDLENGTH_8B

IRDA_WORDLENGTH_9B

IS_IRDA_WORD_LENGTH

28 HAL IWDG Generic Driver

28.1 IWDG Firmware driver registers structures

28.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32f3xx_hal_iwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***
- ***uint32_t Window***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
  Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF

28.1.2 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the stm32f3xx_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

28.2 IWDG Firmware driver API description

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

28.2.1 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 @41KHz (LSI): ~0.1ms / ~25.5s The IWDG timeout may vary due to LSI frequency dispersion. STM32F30x devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy. For more information, please refer to the STM32F3xx Reference manual.

28.2.2 How to use this driver

1. if Window option is disabled
   - 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.
2. if Window option is enabled:
   - Use IWDG using HAL_IWDG_Start() function to enable IWDG downcounter
   - Use IWDG using HAL_IWDG_Init() function to :
     - Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.
     - Configure the IWDG prescaler, reload value and window value.
   - 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_ENABLE_WRITE_ACCESS : Enable write access to IWDG_PR and IWDG_RLR registers
- __HAL_IWDG_DISABLE_WRITE_ACCESS : Disable write access to IWDG_PR and IWDG_RLR registers
- __HAL_IWDG_GET_FLAG: Get the selected IWDG's flag status

28.2.3 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
- Manage Window option
- Initialize the IWDG MSP
- DeInitialize IWDG MSP
- ***HAL_IWDG_Init()***
- ***HAL_IWDG_MspInit()***

28.2.4 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.
- ***HAL_IWDG_Start()***
- ***HAL_IWDG_Refresh()***

28.2.5 Peripheral State functions

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

- ***HAL_IWDG_GetState()***

28.2.6 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

28.2.7 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

28.2.8 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

28.2.9 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

28.2.10 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

28.3 IWDG Firmware driver defines

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

28.3.1 IWDG

IWDG

IWDG CounterWindow Value

IS_IWDG_WINDOW

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_ENABLE_WRITE_ACCESS

Description:

- Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None:

__HAL_IWDG_DISABLE_WRITE_ACCESS

Description:

- Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

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
  - IWDG_FLAG_WVU: Watchdog counter window 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_FLAG_WVU	Watchdog counter window value update Flag
IS_IWDG_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
IS_IWDG_PRESCALER	

IWDG Private Defines

HAL_IWDG_DEFAULT_TIMEOUT

IWDG Registers BitMask

KR_KEY_RELOAD	IWDG Reload Counter Enable
KR_KEY_ENABLE	IWDG Peripheral Enable

KR_KEY_EWA	IWDG KR Write Access Enable
KR_KEY_DWA	IWDG KR Write Access Disable

IS_IWDG_KR

IWDG Reload Value

IS_IWDG_RELOAD

IWDG Window option

IWDG_WINDOW_DISABLE

29 HAL NAND Generic Driver

29.1 NAND Firmware driver registers structures

29.1.1 NAND_IDTypeDef

NAND_IDTypeDef is defined in the stm32f3xx_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***

29.1.2 NAND_AddressTypedef

NAND_AddressTypedef is defined in the stm32f3xx_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

29.1.3 NAND_InfoTypeDef

NAND_InfoTypeDef is defined in the stm32f3xx_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

29.1.4 NAND_HandleTypeDef

NAND_HandleTypeDef is defined in the stm32f3xx_hal_nand.h

Data Fields

- ***FMC_NAND_TypeDef * Instance***
- ***FMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_InfoTypeDef Info***

Field Documentation

- ***FMC_NAND_TypeDef* NAND_HandleTypeDef::Instance***
  Register base address
- ***FMC_NAND_InitTypeDef NAND_HandleTypeDef::Init***
  NAND device control configuration parameters
- ***HAL_LockTypeDef NAND_HandleTypeDef::Lock***
  NAND locking object
- ***__IO HAL_NAND_StateTypeDef NAND_HandleTypeDef::State***
  NAND device access state
- ***NAND_InfoTypeDef NAND_HandleTypeDef::Info***
  NAND characteristic information structure

29.2 NAND Firmware driver API description

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

29.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC/FSMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.
- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page()/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.

29.2.2 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()***

29.2.3 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()*

29.2.4 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()*

29.2.5 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()*

29.2.6 HAL_NAND_Init

Function Name	**HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)**
Function Description	Perform NAND memory Initialization sequence.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • **ComSpace_Timing:** pointer to Common space timing structure • **AttSpace_Timing:** pointer to Attribute space timing structure
Return values	• HAL status

29.2.7 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

29.2.8 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

29.2.9 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

29.2.10 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

29.2.11 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

29.2.12 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

29.2.13 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

29.2.14 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

29.2.15 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

29.2.16 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

29.2.17 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

29.2.18 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

29.2.19 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

29.2.20 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

29.2.21 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

29.2.22 HAL_NAND_ECC_Disable

Function Name	**HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)**
Function Description	Disables dynamically FMC_NAND ECC feature.
Parameters	• **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• HAL status

29.2.23 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

29.2.24 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

29.2.25 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

29.3 NAND Firmware driver defines

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

29.3.1 NAND

NAND

NAND Exported 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_VALID_ADDRESS

NAND_INVALID_ADDRESS

NAND_TIMEOUT_ERROR

NAND_BUSY

NAND_ERROR

NAND_READY

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE	**Description:** • Reset NAND handle state. **Parameters:** • __HANDLE__: specifies the NAND handle. **Return value:** • None:
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	

30 HAL NOR Generic Driver

30.1 NOR Firmware driver registers structures

30.1.1 NOR_IDTypeDef

NOR_IDTypeDef is defined in the stm32f3xx_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.

30.1.2 NOR_CFITypeDef

NOR_CFITypeDef is defined in the stm32f3xx_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***
- ***uint16_t NOR_CFITypeDef::CFI_2***
- ***uint16_t NOR_CFITypeDef::CFI_3***
- ***uint16_t NOR_CFITypeDef::CFI_4***
  Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory.

30.1.3 NOR_HandleTypeDef

NOR_HandleTypeDef is defined in the stm32f3xx_hal_nor.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***

Field Documentation

- ***FMC_NORSRAM_TypeDef* NOR_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* NOR_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef NOR_HandleTypeDef::Init***
  NOR device control configuration parameters
- ***HAL_LockTypeDef NOR_HandleTypeDef::Lock***
  NOR locking object
- ***__IO HAL_NOR_StateTypeDef NOR_HandleTypeDef::State***
  NOR device access state

30.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

30.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()

This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- __NOR_WRITE: NOR memory write data to specified address

30.2.2 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()*

30.2.3 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()*

30.2.4 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()*

30.2.5 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()*

30.2.6 HAL_NOR_Init

Function Name	**HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)**
Function Description	Perform the NOR memory Initialization sequence.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **Timing:** pointer to NOR control timing structure • **ExtTiming:** pointer to NOR extended mode timing structure
Return values	• HAL status

30.2.7 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

30.2.8 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

30.2.9 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

30.2.10 HAL_NOR_MspWait

Function Name	**void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor,**

	uint32_t Timeout)
Function Description	NOR MSP 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

30.2.11 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

30.2.12 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

30.2.13 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

30.2.14 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

30.2.15 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 FMC NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **uwAddress:** NOR memory internal address to read from. • **pData:** pointer to the buffer that receives the data read from the NOR memory. • **uwBufferSize:** number of Half word to read.
Return values	• HAL status

30.2.16 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 FMC NOR memory.
Parameters	• **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • **uwAddress:** NOR memory internal address from which the data • **pData:** pointer to source data buffer. • **uwBufferSize:** number of Half words to write.
Return values	• HAL status
Notes	• Some NOR memory need Address aligned to xx bytes (can be aligned to 64 bytes boundary for example). • The maximum buffer size allowed is NOR memory dependent (can be 64 Bytes max for example).

30.2.17 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

30.2.18 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

30.2.19 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

30.2.20 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

30.2.21 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

30.2.22 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

30.2.23 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

30.3 NOR Firmware driver defines

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

30.3.1 NOR

NOR

NOR Exported Constants

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 Exported Macros

__HAL_NOR_RESET_HANDLE_STATE

Description:

- Reset NOR handle state.

Parameters:

- __HANDLE__: NOR handle

Return value:

- None:

__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:

31 HAL OPAMP Generic Driver

31.1 OPAMP Firmware driver registers structures

31.1.1 OPAMP_InitTypeDef

OPAMP_InitTypeDef is defined in the stm32f3xx_hal_opamp.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t InvertingInput***
- ***uint32_t NonInvertingInput***
- ***uint32_t TimerControlledMuxmode***
- ***uint32_t InvertingInputSecondary***
- ***uint32_t NonInvertingInputSecondary***
- ***uint32_t PgaConnect***
- ***uint32_t PgaGain***
- ***uint32_t UserTrimming***
- ***uint32_t TrimmingValueP***
- ***uint32_t TrimmingValueN***

Field Documentation

- ***uint32_t OPAMP_InitTypeDef::Mode***
  Specifies the OPAMP mode This parameter must be a value of ***OPAMP_Mode*** mode is either Standalone, - Follower or PGA
- ***uint32_t OPAMP_InitTypeDef::InvertingInput***
  Specifies the inverting input in Standalone & Pga modes In Standalone mode: i.e when mode is OPAMP_STANDALONE_MODE This parameter must be a value of ***OPAMP_InvertingInput*** InvertingInput is either VM0 or VM1In PGA mode: i.e when mode is OPAMP_PGA_MODE & in Follower mode i.e when mode is OPAMP_FOLLOWER_MODE This parameter is Not Applicable
- ***uint32_t OPAMP_InitTypeDef::NonInvertingInput***
  Specifies the non inverting input of the opamp: This parameter must be a value of ***OPAMP_NonInvertingInput*** NonInvertingInput is either VP0, VP1, VP2 or VP3
- ***uint32_t OPAMP_InitTypeDef::TimerControlledMuxmode***
  Specifies if the Timer controlled Mux mode is enabled or disabled This parameter must be a value of ***OPAMP_TimerControlledMuxmode***
- ***uint32_t OPAMP_InitTypeDef::InvertingInputSecondary***
  Specifies the inverting input (secondary) of the opamp when TimerControlledMuxmode is enabled i.e. when TimerControlledMuxmode is OPAMP_TIMERCONTROLLEDMUXMODE_ENABLE In Standalone mode: i.e when mode is OPAMP_STANDALONE_MODE This parameter must be a value of ***OPAMP_InvertingInputSecondary*** InvertingInputSecondary is either VM0 or VM1In PGA mode: i.e when mode is OPAMP_PGA_MODE & in Follower mode i.e when mode is OPAMP_FOLLOWER_MODE This parameter is Not Applicable
- ***uint32_t OPAMP_InitTypeDef::NonInvertingInputSecondary***
  Specifies the non inverting input (secondary) of the opamp when TimerControlledMuxmode is enabled i.e. when TimerControlledMuxmode is OPAMP_TIMERCONTROLLEDMUXMODE_ENABLE This parameter must be a value

of ***OPAMP_NonInvertingInputSecondary*** NonInvertingInput is either VP0, VP1, VP2 or VP3

- ***uint32_t OPAMP_InitTypeDef::PgaConnect***
  Specifies the inverting pin in PGA mode i.e. when mode is OPAMP_PGA_MODE This parameter must be a value of ***OPAMP_PgaConnect*** Either: not connected, connected to VM0, connected to VM1 (VM0 or VM1 are typically used for external filtering)
- ***uint32_t OPAMP_InitTypeDef::PgaGain***
  Specifies the gain in PGA mode i.e. when mode is OPAMP_PGA_MODE. This parameter must be a value of ***OPAMP_PgaGain*** (2, 4, 8 or 16)
- ***uint32_t OPAMP_InitTypeDef::UserTrimming***
  Specifies the trimming mode This parameter must be a value of ***OPAMP_UserTrimming*** UserTrimming is either factory or user trimming
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueP***
  Specifies the offset trimming value (PMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueN***
  Specifies the offset trimming value (NMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31

31.1.2 OPAMP_HandleTypeDef

OPAMP_HandleTypeDef is defined in the stm32f3xx_hal_opamp.h

Data Fields

- ***OPAMP_TypeDef * Instance***
- ***OPAMP_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_OPAMP_StateTypeDef State***

Field Documentation

- ***OPAMP_TypeDef* OPAMP_HandleTypeDef::Instance***
  OPAMP instance's registers base address
- ***OPAMP_InitTypeDef OPAMP_HandleTypeDef::Init***
  OPAMP required parameters
- ***HAL_StatusTypeDef OPAMP_HandleTypeDef::Status***
  OPAMP peripheral status
- ***HAL_LockTypeDef OPAMP_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_OPAMP_StateTypeDef OPAMP_HandleTypeDef::State***
  OPAMP communication state

31.2 OPAMP Firmware driver API description

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

31.2.1 OPAMP Peripheral Features

The device integrates up to 4 operational amplifiers OPAMP1, OPAMP2, OPAMP3 and OPAMP4:

1. The OPAMP(s) provides several exclusive running modes.
   - Standalone mode
   - Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
   - Follower mode
2. The OPAMP(s) provide(s) calibration capabilities.
   - Calibration aims at correcting some offset for running mode.
   - The OPAMP uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   - The user defined settings can be figured out using self calibration handled by HAL_OPAMP_SelfCalibrate, HAL_OPAMPEx_SelfCalibrateAll
   - HAL_OPAMP_SelfCalibrate:
   - Runs automatically the calibration in 2 steps. (90% of VDDA for NMOS transistors, 10% of VDDA for PMOS transistors). (As OPAMP is Rail-to-rail input/output, these 2 steps calibration is appropriate and enough in most cases).
   - Enables the user trimming mode
   - Updates the init structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)
   - for STM32F3 devices having 2 or 4 OPAMPs HAL_OPAMPEx_SelfCalibrateAll runs calibration of 2 or 4 OPAMPs in parallel.
3. For any running mode, an additional Timer-controlled Mux (multiplexer) mode can be set on top.
   - Timer-controlled Mux mode allows Automatic switching between inverting and non-inverting input.
   - Hence on top of defaults (primary) inverting and non-inverting inputs, the user shall select secondary inverting and non inverting inputs.
   - TIM1 CC6 provides the alternate switching tempo between defaults (primary) and secondary inputs.
4. Running mode: Standalone mode
   - Gain is set externally (gain depends on external loads).
   - Follower mode also possible externally by connecting the inverting input to the output.
5. Running mode: Follower mode
   - No Inverting Input is connected.
6. Running mode: Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
   - The OPAMP(s) output(s) can be internally connected to resistor feedback output.
   - OPAMP gain is either 2, 4, 8 or 16.
7. The OPAMPs non inverting input (both default and secondary) can be selected among the list shown in *Table 23: "OPAMPs inverting/non-inverting inputs for STM32F3 devices"* .
8. The OPAMPs non inverting input (both default and secondary) can be selected among the list shown in *Table 24: "OPAMP outputs for STM32F3 devices"*.

Table 23: OPAMPs inverting/non-inverting inputs for STM32F3 devices

	HAL parameter name	OPAMP1	OPAMP2	OPAMP3	OPAMP4
Inverting inputs[(1)]	Non connected VM0 VM1	X PC5 PA3	X PC5 PA5	X PB10 PB2	X PB10 PD8
Non-inverting inputs	VP0 VP1 VP2 VP3	PA1 PA7 PA3 PA5	PA7 PD14 PB0 PB14	PB0 PB13 PA1 PA5	PB13 PD11 PA4 PB11

Notes:

[(1)]NA in follower mode.

Table 24: OPAMP outputs for STM32F3 devices

	OPAMP1	OPAMP2	OPAMP3	OPAMP4
Output	PA2	PA6	PB1	PB12

31.2.2 How to use this driver

Calibration

To run the opamp calibration self calibration:

1. Start calibration using HAL_OPAMP_SelfCalibrate. Store the calibration results.

Running mode

To use the opamp, perform the following steps:

1. Fill in the HAL_OPAMP_MspInit() to
   – Configure the opamp input AND output in analog mode using HAL_GPIO_Init() to map the opamp output to the GPIO pin.
2. Configure the opamp using HAL_OPAMP_Init() function:
   – Select the mode
   – Select the inverting input
   – Select the non-inverting input
   – Select if the Timer controlled Mux mode is enabled/disabled
   – If the Timer controlled Mux mode is enabled, select the secondary inverting input
   – If the Timer controlled Mux mode is enabled, Select the secondary non-inverting input
   – If PGA mode is enabled, Select if inverting input is connected.
   – Select either factory or user defined trimming mode.
   – If the user defined trimming mode is enabled, select PMOS & NMOS trimming values (typ. settings returned by HAL_OPAMP_SelfCalibrate function).
3. Enable the opamp using HAL_OPAMP_Start() function.
4. Disable the opamp using HAL_OPAMP_Stop() function.

5. Lock the opamp in running mode using HAL_OPAMP_Lock() function. From then The configuration can only be modified after HW reset.

Running mode: change of configuration while OPAMP ON

To Re-configure OPAMP when OPAMP is ON (change on the fly)

1. If needed, Fill in the HAL_OPAMP_MspInit()
   – This is the case for instance if you wish to use new OPAMP I/O
2. Configure the opamp using HAL_OPAMP_Init() function:
   – As in configure case, selects first the parameters you wish to modify.

31.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- ***HAL_OPAMP_Init()***
- ***HAL_OPAMP_DeInit()***
- ***HAL_OPAMP_MspInit()***
- ***HAL_OPAMP_MspDeInit()***

31.2.4 IO operation functions

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

- ***HAL_OPAMP_Start()***
- ***HAL_OPAMP_Stop()***
- ***HAL_OPAMP_SelfCalibrate()***

31.2.5 Peripheral Control functions

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

- ***HAL_OPAMP_Lock()***

31.2.6 Peripheral State functions

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

- ***HAL_OPAMP_GetState()***
- ***HAL_OPAMP_GetTrimOffset()***

31.2.7 HAL_OPAMP_Init

Function Name	**HAL_StatusTypeDef HAL_OPAMP_Init (OPAMP_HandleTypeDef * hopamp)**
Function Description	Initializes the OPAMP according to the specified parameters in the OPAMP_InitTypeDef and create the associated handle.
Parameters	• **hopamp:** OPAMP handle
Return values	• HAL status

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

31.2.8 HAL_OPAMP_DeInit

Function Name	**HAL_StatusTypeDef HAL_OPAMP_DeInit (OPAMP_HandleTypeDef * hopamp)**
Function Description	DeInitializes the OPAMP peripheral.
Parameters	• **hopamp:** OPAMP handle
Return values	• HAL status
Notes	• Deinitialization can't be performed if the OPAMP configuration is locked. To unlock the configuration, perform a system reset.

31.2.9 HAL_OPAMP_MspInit

Function Name	**void HAL_OPAMP_MspInit (OPAMP_HandleTypeDef * hopamp)**
Function Description	Initializes the OPAMP MSP.
Parameters	• **hopamp:** OPAMP handle
Return values	• None

31.2.10 HAL_OPAMP_MspDeInit

Function Name	**void HAL_OPAMP_MspDeInit (OPAMP_HandleTypeDef * hopamp)**
Function Description	DeInitializes OPAMP MSP.
Parameters	• **hopamp:** OPAMP handle
Return values	• None

31.2.11 HAL_OPAMP_Start

Function Name	**HAL_StatusTypeDef HAL_OPAMP_Start (OPAMP_HandleTypeDef * hopamp)**
Function Description	Start the opamp.
Parameters	• **hopamp:** OPAMP handle
Return values	• HAL status

31.2.12 HAL_OPAMP_Stop

Function Name	**HAL_StatusTypeDef HAL_OPAMP_Stop (OPAMP_HandleTypeDef * hopamp)**
Function Description	Stop the opamp.
Parameters	• **hopamp:** OPAMP handle
Return values	• HAL status

31.2.13 HAL_OPAMP_SelfCalibrate

Function Name	**HAL_StatusTypeDef HAL_OPAMP_SelfCalibrate (OPAMP_HandleTypeDef * hopamp)**
Function Description	Run the self calibration of one OPAMP.
Parameters	• **hopamp:** handle
Return values	• Updated offset trimming values (PMOS & NMOS), user trimming is enabled • HAL status
Notes	• Calibration runs about 25 ms.

31.2.14 HAL_OPAMP_Lock

Function Name	**HAL_StatusTypeDef HAL_OPAMP_Lock (OPAMP_HandleTypeDef * hopamp)**
Function Description	Lock the selected opamp configuration.
Parameters	• **hopamp:** OPAMP handle
Return values	• HAL status

31.2.15 HAL_OPAMP_GetState

Function Name	**HAL_OPAMP_StateTypeDef HAL_OPAMP_GetState (OPAMP_HandleTypeDef * hopamp)**
Function Description	Return the OPAMP state.
Parameters	• **hopamp:** OPAMP handle
Return values	• HAL state

31.2.16 HAL_OPAMP_GetTrimOffset

Function Name	**OPAMP_TrimmingValueTypeDef HAL_OPAMP_GetTrimOffset (OPAMP_HandleTypeDef * hopamp, uint32_t trimmingoffset)**
Function Description	Return the OPAMP factory trimming value.
Parameters	• **hopamp:** OPAMP handle • **trimmingoffset:** Trimming offset (P or N)

Return values	•	Trimming value (P or N): range: 0->31 or OPAMP_FACTORYTRIMMING_DUMMY if trimming value is not available

31.3 OPAMP Firmware driver defines

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

31.3.1 OPAMP

OPAMP

OPAMP CSR init register Mask

OPAMP_CSR_UPDATE_PARAMETERS_INIT_MASK

OPAMP Exported Macros

__HAL_OPAMP_RESET_HANDLE_STATE	**Description:** • Reset OPAMP handle state. **Parameters:** • __HANDLE__: OPAMP handle. **Return value:** • None:

OPAMP Factory Trimming

OPAMP_FACTORYTRIMMING_DUMMY	Dummy trimming value
OPAMP_FACTORYTRIMMING_N	Offset trimming N
OPAMP_FACTORYTRIMMING_P	Offset trimming P
IS_OPAMP_FACTORYTRIMMING	

OPAMP Input

OPAMP_INPUT_INVERTING	Inverting input
OPAMP_INPUT_NONINVERTING	Non inverting input
IS_OPAMP_INPUT	

OPAMP Inverting Input

IOPAMP_INVERTINGINPUT_VM0	inverting input connected to VM0
IOPAMP_INVERTINGINPUT_VM1	inverting input connected to VM1
IS_OPAMP_INVERTING_INPUT	

OPAMP Inverting Input Secondary

OPAMP_SEC_INVERTINGINPUT_VM0	VM0 (PC5 for OPAMP1 and OPAMP2, PB10 for OPAMP3 and OPAMP4) connected to OPAMPx inverting input
OPAMP_SEC_INVERTINGINPUT_VM1	VM1 (PA3 for OPAMP1, PA5 for OPAMP2, PB2 for OPAMP3, PD8 for OPAMP4) connected to OPAMPx inverting input
IS_OPAMP_SEC_INVERTINGINPUT	

OPAMP Mode

OPAMP_STANDALONE_MODE	standalone mode
OPAMP_PGA_MODE	PGA mode
OPAMP_FOLLOWER_MODE	follower mode

IS_OPAMP_FUNCTIONAL_NORMALMODE

OPAMP Non Inverting Input

OPAMP_NONINVERTINGINPUT_VP0	VP0 (PA1 for OPAMP1, PA7 for OPAMP2, PB0 for OPAMP3, PB13 for OPAMP4) connected to OPAMPx non inverting input
OPAMP_NONINVERTINGINPUT_VP1	VP1 (PA7 for OPAMP1, PD14 for OPAMP2, PB13 for OPAMP3, PD11 for OPAMP4) connected to OPAMPx non inverting input
OPAMP_NONINVERTINGINPUT_VP2	VP2 (PA3 for OPAMP1, PB0 for OPAMP2, PA1 for OPAMP3, PA4 for OPAMP4) connected to OPAMPx non inverting input
OPAMP_NONINVERTINGINPUT_VP3	vp3 (PA5 for OPAMP1, PB14 for OPAMP2, PA5 for OPAMP3, PB11 for OPAMP4) connected to OPAMPx non inverting input

IS_OPAMP_NONINVERTING_INPUT

OPAMP Non Inverting Input Secondary

OPAMP_SEC_NONINVERTINGINPUT_VP0	VP0 (PA1 for OPAMP1, PA7 for OPAMP2, PB0 for OPAMP3, PB13 for OPAMP4) connected to OPAMPx non inverting input
OPAMP_SEC_NONINVERTINGINPUT_VP1	VP1 (PA7 for OPAMP1, PD14 for OPAMP2, PB13 for OPAMP3, PD11 for OPAMP4) connected to OPAMPx non inverting input
OPAMP_SEC_NONINVERTINGINPUT_VP2	VP2 (PA3 for OPAMP1, PB0 for OPAMP2, PA1 for OPAMP3, PA4 for OPAMP4) connected to OPAMPx non inverting input
OPAMP_SEC_NONINVERTINGINPUT_VP3	VP3 (PA5 for OPAMP1, PB14 for OPAMP2, PA5 for OPAMP3, PB11 for OPAMP4) connected to OPAMPx non inverting input

IS_OPAMP_SEC_NONINVERTINGINPUT

OPAMP Pga Connect

OPAMP_PGACONNECT_NO	In PGA mode, the non inverting input is not connected
OPAMP_PGACONNECT_VM0	In PGA mode, the non inverting input is connected to VM0
OPAMP_PGACONNECT_VM1	In PGA mode, the non inverting input is connected to VM1

IS_OPAMP_PGACONNECT

OPAMP Pga Gain

OPAMP_PGA_GAIN_2	PGA gain = 2
OPAMP_PGA_GAIN_4	PGA gain = 4
OPAMP_PGA_GAIN_8	PGA gain = 8
OPAMP_PGA_GAIN_16	PGA gain = 16
IS_OPAMP_PGA_GAIN	

OPAMP Private Define

OPAMP_CSR_RESET_VALUE

OPAMP Timer Controlled Mux mode

OPAMP_TIMERCONTROLLEDMUXMODE_DISABLE	Timer controlled Mux mode disabled
OPAMP_TIMERCONTROLLEDMUXMODE_ENABLE	Timer controlled Mux mode enabled
IS_OPAMP_TIMERCONTROLLED_MUXMODE	

OPAMP Trimming Value

IS_OPAMP_TRIMMINGVALUE

OPAMP User Trimming

OPAMP_TRIMMING_FACTORY	Factory trimming
OPAMP_TRIMMING_USER	User trimming
IS_OPAMP_TRIMMING	
OPAMP_VREF_NOTCONNECTEDTO_ADC	VREF not connected to ADC
OPAMP_VREF_CONNECTEDTO_ADC	VREF not connected to ADC
IS_OPAMP_ALLOPAMPVREF_CONNECT	

OPAMP VREF

OPAMP_VREF_3VDDA	OPMAP Vref = 3.3% VDDA
OPAMP_VREF_10VDDA	OPMAP Vref = 10% VDDA
OPAMP_VREF_50VDDA	OPMAP Vref = 50% VDDA
OPAMP_VREF_90VDDA	OPMAP Vref = 90% VDDA
IS_OPAMP_VREF	

32 HAL OPAMP Extension Driver

32.1 OPAMPEx Firmware driver API description

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

32.1.1 HAL_OPAMPEx_SelfCalibrateAll

Function Name	**HAL_StatusTypeDef HAL_OPAMPEx_SelfCalibrateAll (OPAMP_HandleTypeDef * hopamp1, OPAMP_HandleTypeDef * hopamp2, OPAMP_HandleTypeDef * hopamp3, OPAMP_HandleTypeDef * hopamp4)**
Function Description	Run the self calibration of 4 OPAMPs in parallel.
Parameters	• **hopamp1:** handle • **hopamp2:** handle • **hopamp3:** handle • **hopamp4:** handle
Return values	• HAL status
Notes	• Updated offset trimming values (PMOS & NMOS), user trimming is enabled • Calibration runs about 25 ms.

32.2 OPAMPEx Firmware driver defines

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

32.2.1 OPAMPEx

OPAMPEx

33 HAL PCCARD Generic Driver

33.1 PCCARD Firmware driver registers structures

33.1.1 PCCARD_HandleTypeDef

PCCARD_HandleTypeDef is defined in the stm32f3xx_hal_pccard.h

Data Fields

- ***FMC_PCCARD_TypeDef * Instance***
- ***FMC_PCCARD_InitTypeDef Init***
- ***__IO HAL_PCCARD_StateTypeDef State***
- ***HAL_LockTypeDef Lock***

Field Documentation

- ***FMC_PCCARD_TypeDef* PCCARD_HandleTypeDef::Instance***
  Register base address for PCCARD device
- ***FMC_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

33.2 PCCARD Firmware driver API description

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

33.2.1 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 FMC 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.

33.2.2 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()***

33.2.3 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()***

33.2.4 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()***

33.2.5 HAL_PCCARD_Init

Function Name	**HAL_StatusTypeDef HAL_PCCARD_Init (PCCARD_HandleTypeDef * hpccard, FMC_NAND_PCC_TimingTypeDef * ComSpaceTiming, FMC_NAND_PCC_TimingTypeDef * AttSpaceTiming, FMC_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

33.2.6 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

33.2.7 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

33.2.8 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

33.2.9 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

33.2.10 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

33.2.11 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

33.2.12 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

33.2.13 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

33.2.14 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

33.2.15 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

33.2.16 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

33.2.17 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

33.2.18 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

33.3 PCCARD Firmware driver defines

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

33.3.1 PCCARD

PCCARD

PCCARD Compact Flash ATA Commands

CF_READ_SECTOR_CMD

CF_WRITE_SECTOR_CMD

CF_ERASE_SECTOR_CMD

CF_IDENTIFY_CMD

PCCARD Compact Flash ATA Registers

CF_DATA	Data register

CF_SECTOR_COUNT	Sector Count register
CF_SECTOR_NUMBER	Sector Number register
CF_CYLINDER_LOW	Cylinder low register
CF_CYLINDER_HIGH	Cylinder high register
CF_CARD_HEAD	Card/Head register
CF_STATUS_CMD	Status(read)/Command(write) register
CF_STATUS_CMD_ALTERNATE	Alternate Status(read)/Command(write) register
CF_COMMON_DATA_AREA	Start of data area (for Common access only!)

PCCARD Compact Flash Characteristics

CF_DEVICE_ADDRESS	
CF_ATTRIBUTE_SPACE_ADDRESS	Attribute space size to @0x9BFF FFFF
CF_COMMON_SPACE_ADDRESS	Common space size to @0x93FF FFFF
CF_IO_SPACE_ADDRESS	IO space size to @0x9FFF FFFF
CF_IO_SPACE_PRIMARY_ADDR	IO space size to @0x9FFF FFFF

PCCARD_Compact_Flash_Sector_Size

CF_SECTOR_SIZE

PCCARD Compact Flash Status

CF_TIMEOUT_ERROR

CF_BUSY

CF_PROGR

CF_READY

PCCARD Exported Macros

__HAL_PCCARD_RESET_HANDLE_STATE

Description:

- Reset PCCARD handle state.

Parameters:

- __HANDLE__: specifies the PCCARD handle.

Return value:

- None:

34 HAL PCD Generic Driver

34.1 PCD Firmware driver registers structures

34.1.1 PCD_InitTypeDef

PCD_InitTypeDef is defined in the stm32f3xx_hal_pcd.h

Data Fields

- ***uint32_t dev_endpoints***
- ***uint32_t speed***
- ***uint32_t ep0_mps***
- ***uint32_t phy_itface***
- ***uint32_t Sof_enable***
- ***uint32_t low_power_enable***
- ***uint32_t lpm_enable***
- ***uint32_t battery_charging_enable***

Field Documentation

- ***uint32_t PCD_InitTypeDef::dev_endpoints***
  Device Endpoints number. This parameter depends on the used USB core. This parameter must be a number between Min_Data = 1 and Max_Data = 15
- ***uint32_t PCD_InitTypeDef::speed***
  USB Core speed. This parameter can be any value of ***PCD_Core_Speed***
- ***uint32_t PCD_InitTypeDef::ep0_mps***
  Set the Endpoint 0 Max Packet size. This parameter can be any value of ***PCD_EP0_MPS***
- ***uint32_t PCD_InitTypeDef::phy_itface***
  Select the used PHY interface. This parameter can be any value of ***PCD_Core_PHY***
- ***uint32_t PCD_InitTypeDef::Sof_enable***
  Enable or disable the output of the SOF signal. This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_InitTypeDef::low_power_enable***
  Enable or disable Low Power mode This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_InitTypeDef::lpm_enable***
  Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32_t PCD_InitTypeDef::battery_charging_enable***
  Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE

34.1.2 PCD_EPTypeDef

PCD_EPTypeDef is defined in the stm32f3xx_hal_pcd.h

Data Fields

- ***uint8_t num***
- ***uint8_t is_in***
- ***uint8_t is_stall***
- ***uint8_t type***
- ***uint16_t pmaadress***
- ***uint16_t pmaaddr0***
- ***uint16_t pmaaddr1***
- ***uint8_t doublebuffer***
- ***uint32_t maxpacket***
- ***uint8_t * xfer_buff***
- ***uint32_t xfer_len***
- ***uint32_t xfer_count***

Field Documentation

- ***uint8_t PCD_EPTypeDef::num***
  Endpoint number This parameter must be a number between Min_Data = 1 and Max_Data = 15
- ***uint8_t PCD_EPTypeDef::is_in***
  Endpoint direction This parameter must be a number between Min_Data = 0 and Max_Data = 1
- ***uint8_t PCD_EPTypeDef::is_stall***
  Endpoint stall condition This parameter must be a number between Min_Data = 0 and Max_Data = 1
- ***uint8_t PCD_EPTypeDef::type***
  Endpoint type This parameter can be any value of ***PCD_EP_Type***
- ***uint16_t PCD_EPTypeDef::pmaadress***
  PMA Address This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- ***uint16_t PCD_EPTypeDef::pmaaddr0***
  PMA Address0 This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- ***uint16_t PCD_EPTypeDef::pmaaddr1***
  PMA Address1 This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- ***uint8_t PCD_EPTypeDef::doublebuffer***
  Double buffer enable This parameter can be 0 or 1
- ***uint32_t PCD_EPTypeDef::maxpacket***
  Endpoint Max packet size This parameter must be a number between Min_Data = 0 and Max_Data = 64KB
- ***uint8_t* PCD_EPTypeDef::xfer_buff***
  Pointer to transfer buffer
- ***uint32_t PCD_EPTypeDef::xfer_len***
  Current transfer length
- ***uint32_t PCD_EPTypeDef::xfer_count***
  Partial transfer length in case of multi packet transfer

34.1.3 PCD_HandleTypeDef

PCD_HandleTypeDef is defined in the stm32f3xx_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
- ***PCD_EPTypeDef PCD_HandleTypeDef::IN_ep[8]***
  IN endpoint parameters
- ***PCD_EPTypeDef PCD_HandleTypeDef::OUT_ep[8]***
  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

34.2 PCD Firmware driver API description

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

34.2.1 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
      – __USB_CLK_ENABLE();
   b. Initialize the related GPIO clocks
   c. Configure PCD pin-out
   d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:

a. hpcd.pData = pdev;

6. Enable HCD transmission and reception:

a. HAL_PCD_Start();

34.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- *HAL_PCD_Init()*
- *HAL_PCD_DeInit()*
- *HAL_PCD_MspInit()*
- *HAL_PCD_MspDeInit()*

34.2.3 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()*

34.2.4 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()*

34.2.5 Peripheral State functions

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

- *HAL_PCD_GetState()*

34.2.6 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

34.2.7 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

34.2.8 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

34.2.9 HAL_PCD_MspDeInit

Function Name	**void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)**
Function Description	DeInitializes PCD MSP.
Parameters	• **hpcd:** PCD handle
Return values	• None

34.2.10 HAL_PCD_Start

Function Name	**HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef ***

	hpcd)
Function Description	Start The USB OTG Device.
Parameters	• **hpcd:** PCD handle
Return values	• HAL status

34.2.11 HAL_PCD_Stop

Function Name	**HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)**
Function Description	Stop The USB OTG Device.
Parameters	• **hpcd:** PCD handle
Return values	• HAL status

34.2.12 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

34.2.13 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

34.2.14 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

34.2.15 HAL_PCD_SetupStageCallback

Function Name	**void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Setup stage callback.
Parameters	• **hpcd:** ppp handle
Return values	• None

34.2.16 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

34.2.17 HAL_PCD_ResetCallback

Function Name	**void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)**
Function Description	USB Reset callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

34.2.18 HAL_PCD_SuspendCallback

Function Name	**void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Suspend event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

34.2.19 HAL_PCD_ResumeCallback

Function Name	**void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Resume event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

34.2.20 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

34.2.21 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

34.2.22 HAL_PCD_ConnectCallback

Function Name	**void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Connection event callbacks.
Parameters	• **hpcd:** PCD handle
Return values	• None

34.2.23 HAL_PCD_DisconnectCallback

Function Name	**void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)**
Function Description	Disconnection event callbacks.
Parameters	• **hpcd:** ppp handle
Return values	• None

34.2.24 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

34.2.25 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

34.2.26 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

34.2.27 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 packert size • **ep_type:** endpoint type
Return values	• HAL status

34.2.28 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

34.2.29 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

34.2.30 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

34.2.31 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

34.2.32 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

34.2.33 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

34.2.34 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

34.2.35 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	• status

34.2.36 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	• status

34.2.37 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

34.2.38 PCD_WritePMA

Function Name	**void PCD_WritePMA (USB_TypeDef * USBx, uint8_t * pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)**
Function Description	Copy a buffer from user memory area to packet memory area (PMA)
Parameters	• **USBx:** USB peripheral instance register address. • **pbUsrBuf:** pointer to user memory area. • **wPMABufAddr:** address into PMA. • **wNBytes:** no. of bytes to be copied.
Return values	• None

34.2.39 PCD_ReadPMA

Function Name	**void PCD_ReadPMA (USB_TypeDef * USBx, uint8_t * pbUsrBuf, uint16_t wPMABufAddr, uint16_t wNBytes)**
Function Description	Copy a buffer from user memory area to packet memory area (PMA)
Parameters	• **USBx:** USB peripheral instance register address. • **pbUsrBuf:** pointer to user memory area. • **wPMABufAddr:** address into PMA. • **wNBytes:** no. of bytes to be copied.
Return values	• None

34.3 PCD Firmware driver defines

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

34.3.1 PCD

PCD

PCD Core PHY

PCD_PHY_EMBEDDED

PCD Core Speed

PCD_SPEED_HIGH

PCD_SPEED_FULL

PCD_ENDP_Type

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD_SNG_BUF

PCD_DBL_BUF

IS_PCD_ALL_INSTANCE

PCD EP0 MPS

DEP0CTL_MPS_64

DEP0CTL_MPS_32

DEP0CTL_MPS_16

DEP0CTL_MPS_8

PCD_EP0MPS_64

PCD_EP0MPS_32

PCD_EP0MPS_16

PCD_EP0MPS_08

PCD EP Type

PCD_EP_TYPE_CTRL

PCD_EP_TYPE_ISOC

PCD_EP_TYPE_BULK

PCD_EP_TYPE_INTR

PCD Exported Macros

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_USB_EXTI_ENABLE_IT

__HAL_USB_EXTI_DISABLE_IT

__HAL_USB_EXTI_GENERATE_SWIT

__HAL_USB_EXTI_GET_FLAG

__HAL_USB_EXTI_CLEAR_FLAG

__HAL_USB_EXTI_SET_RISING_EDGE_TRIGGER

__HAL_USB_EXTI_SET_FALLING_EDGE_TRIGGER

__HAL_USB_EXTI_SET_FALLINGRISING_TRIGGER

PCD Private Define

BTABLE_ADDRESS

PCD Private Macros

PCD_SET_ENDPOINT

PCD_GET_ENDPOINT

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_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

35 HAL PCD Extension Driver

35.1 PCDEx Firmware driver API description

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

35.1.1 Peripheral extended features methods

- *HAL_PCDEx_PMAConfig()*
- *HAL_PCDEx_SetConnectionState()*

35.1.2 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:** PCD handle • **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	• : status

35.1.3 HAL_PCDEx_SetConnectionState

Function Name	**__weak void HAL_PCDEx_SetConnectionState (PCD_HandleTypeDef * hpcd, uint8_t state)**
Function Description	Software Device Connection.
Parameters	• **hpcd:** PCD handle • **state:** Device state
Return values	• None

35.2 PCDEx Firmware driver defines

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

35.2.1 PCDEx

PCDEx

PCD Extended Exported Macros

PCD_EP_TX_ADDRESS

Description:

- Gets address in an endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None:

PCD_EP_TX_CNT

PCD_EP_RX_ADDRESS

PCD_EP_RX_CNT

PCD_SET_EP_RX_CNT

36 HAL PWR Generic Driver

36.1 PWR Firmware driver API description

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

36.1.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses. To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the __PWR_CLK_ENABLE() macro.
- Enable access to RTC domain using the HAL_PWR_EnableBkUpAccess() function.

36.1.2 Peripheral Control functions

WakeUp pin configuration

- WakeUp pin is used to wakeup the system from Standby mode. This pin is forced in input pull down configuration and is active on rising edges.
- There are up to three WakeUp pins:
  - WakeUp Pin 1 on PA.00.
  - WakeUp Pin 2 on PC.13 (STM32F303xC, STM32F303xE only).
  - WakeUp Pin 3 on PE.06.

Main and Backup Regulators configuration

- When the backup domain is supplied by VDD (analog switch connected to VDD) the backup SRAM is powered from VDD which replaces the VBAT power supply to save battery life.
- The backup SRAM is not mass erased by an tamper event. It is read protected to prevent confidential data, such as cryptographic private key, from being accessed. The backup SRAM can be erased only through the Flash interface when a protection level change from level 1 to level 0 is requested. Refer to the description of Read protection (RDP) in the Flash programming manual. Refer to the datasheets for more details.

Low Power modes configuration

The devices feature 3 low-power modes:

- Sleep mode: Cortex-M4 core stopped, peripherals kept running.
- Stop mode: all clocks are stopped, regulator running, regulator in low power mode
- Standby mode: 1.2V domain powered off (mode not available on STM32F3x8 devices).

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:
  - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

Stop mode

In Stop mode, all clocks in the 1.8V domain are stopped, the PLL, the HSI, and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. The voltage regulator can be configured either in normal or low-power mode. To minimize the consumption.

- Entry: The Stop mode is entered using the HAL_PWR_EnterSTOPMode(PWR_MAINREGULATOR_ON, PWR_STOPENTRY_WFI) function with:
  - Main regulator ON.
  - Low Power regulator ON.
  - PWR_STOPENTRY_WFI: enter STOP mode with WFI instruction
  - PWR_STOPENTRY_WFE: enter STOP mode with WFE instruction
- Exit:
  - Any EXTI Line (Internal or External) configured in Interrupt/Event mode.
  - Some specific communication peripherals (CEC, USART, I2C) interrupts, when programmed in wakeup mode (the peripheral must be programmed in wakeup mode and the corresponding interrupt vector must be enabled in the NVIC)

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M4 deep sleep mode, with the voltage regulator disabled. The 1.8V 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 the RTC registers, RTC backup registers, backup SRAM and Standby circuitry. The voltage regulator is OFF.

- Entry:
  - The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function.
- Exit:
  - WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event, a time-stamp event, or a comparator event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop and Standby modes
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL_RTC_SetAlarm_IT() function.

  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL_RTC_SetTimeStamp_IT() or HAL_RTC_SetTamper_IT() functions.
  - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL_RTC_SetWakeUpTimer_IT() function.
- Comparator auto-wakeup (AWU) from the Stop mode
  - To wake up from the Stop mode with a comparator wakeup event, it is necessary to:
    - Configure the EXTI Line associated with the comparator (example EXTI Line 22 for comparator 2) to be sensitive to to the selected edges (falling, rising or falling and rising) (Interrupt or Event modes) using the EXTI_Init() function.
    - Configure the comparator to generate the event.
- ***HAL_PWR_EnableWakeUpPin()***
- ***HAL_PWR_DisableWakeUpPin()***
- ***HAL_PWR_EnterSLEEPMode()***
- ***HAL_PWR_EnterSTOPMode()***
- ***HAL_PWR_EnterSTANDBYMode()***
- ***HAL_PWR_EnableBkUpAccess()***
- ***HAL_PWR_DisableBkUpAccess()***

36.1.3 HAL_PWR_EnableBkUpAccess

Function Name	**void HAL_PWR_EnableBkUpAccess (void)**
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).
Return values	• None
Notes	• If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

36.1.4 HAL_PWR_DisableBkUpAccess

Function Name	**void HAL_PWR_DisableBkUpAccess (void)**
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).
Return values	• None
Notes	• If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

36.1.5 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, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3
Return values	•	None

36.1.6 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, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3
Return values	• None

36.1.7 HAL_PWR_EnterSLEEPMode

Function Name	**void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)**
Function Description	Enters Sleep mode.
Parameters	• **Regulator:** Specifies the regulator state in SLEEP mode. This parameter can be one of the following values: PWR_MAINREGULATOR_ON: SLEEP mode with regulator ON PWR_LOWPOWERREGULATOR_ON: SLEEP mode with low power regulator ON • **SLEEPEntry:** Specifies if SLEEP mode is entered with WFI or WFE instruction. 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.

36.1.8 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 issuing an interrupt or a wakeup event, the 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.

36.1.9 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)RTC alternate function pins if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out.WKUP pins if enabled.

36.1.10 HAL_PWR_EnableBkUpAccess

Function Name	**void HAL_PWR_EnableBkUpAccess (void)**
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).
Return values	• None
Notes	• If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

36.1.11 HAL_PWR_DisableBkUpAccess

Function Name	**void HAL_PWR_DisableBkUpAccess (void)**
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).
Return values	• None
Notes	• If the HSE divided by 32 is used as the RTC clock, the Backup Domain Access should be kept enabled.

36.2 PWR Firmware driver defines

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

36.2.1 PWR

PWR

PWR Alias Exported Constants

PWR_OFFSET

CR_OFFSET

DBP_BitNumber

CR_DBP_BB

PVDE_BitNumber

CR_PVDE_BB

CSR_OFFSET

EWUP1_BitNumber

CSR_EWUP1_BB

EWUP2_BitNumber

CSR_EWUP2_BB

EWUP3_BitNumber

CSR_EWUP3_BB

PWR Exported Macro

__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 (Alarm A or Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup. 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.
  - PWR_FLAG_VREFINTRDY: This flag indicates that the internal reference voltage VREFINT is

ready.

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

PWR Flag

PWR_FLAG_WU

PWR_FLAG_SB

PWR_FLAG_PVDO

PWR_FLAG_VREFINTRDY

IS_PWR_GET_FLAG

PWR Regulator state in STOP mode

PWR_MAINREGULATOR_ON

PWR_LOWPOWERREGULATOR_ON

IS_PWR_REGULATOR

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

PWR_SLEEPENTRY_WFE

IS_PWR_SLEEP_ENTRY

PWR STOP mode entry

PWR_STOPENTRY_WFI

PWR_STOPENTRY_WFE

IS_PWR_STOP_ENTRY

PWR WakeUp Pins

PWR_WAKEUP_PIN1

PWR_WAKEUP_PIN2

PWR_WAKEUP_PIN3

IS_PWR_WAKEUP_PIN

37 HAL PWR Extension Driver

37.1 PWREx Firmware driver registers structures

37.1.1 PWR_PVDTypeDef

PWR_PVDTypeDef is defined in the stm32f3xx_hal_pwr_ex.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 ***PWREx_PVD_detection_level***
- ***uint32_t PWR_PVDTypeDef::Mode***
  Mode: Specifies the operating mode for the selected pins. This parameter can be a value of ***PWREx_PVD_Mode***

37.2 PWREx Firmware driver API description

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

37.2.1 Peripheral Extended control functions

PVD configuration (present on all other devices than STM32F3x8 devices)

- 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. PVD is not available on STM32F3x8 Product Line

Voltage regulator

- The voltage regulator is always enabled after Reset. It works in three different modes: In Run mode, the regulator supplies full power to the 1.8V domain (core, memories and digital peripherals). In Stop mode, the regulator supplies low power to the 1.8V domain, preserving contents of registers and SRAM. In Stop mode, the regulator is powered off. The contents of the registers and SRAM are lost except for the Standby circuitry and the Backup Domain. Note: In the STM32F3x8xx devices, the voltage regulator is bypassed and the microcontroller must be powered from a nominal VDD = 1.8V +/-8% voltage.

- 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.

SDADC power configuration

- On STM32F373xC/STM32F378xx devices, there are up to 3 SDADC instances that can be enabled/disabled. This is done by calling the following functions:
  void HAL_PWREx_EnableSDADCAnalog(uint32_t Analogx);
  void HAL_PWREx_DisableSDADCAnalog(uint32_t Analogx);
- ***HAL_PWR_PVDConfig()***
- ***HAL_PWR_EnablePVD()***
- ***HAL_PWR_DisablePVD()***
- ***HAL_PWR_PVD_IRQHandler()***
- ***HAL_PWR_PVDCallback()***

37.2.2 HAL_PWR_PVDConfig

Function Name	**void HAL_PWR_PVDConfig (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.

37.2.3 HAL_PWR_EnablePVD

Function Name	**void HAL_PWR_EnablePVD (void)**
Function Description	Enables the Power Voltage Detector(PVD).
Return values	• None

37.2.4 HAL_PWR_DisablePVD

Function Name	**void HAL_PWR_DisablePVD (void)**
Function Description	Disables the Power Voltage Detector(PVD).
Return values	• None

37.2.5 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().

37.2.6 HAL_PWR_PVDCallback

Function Name	**void HAL_PWR_PVDCallback (void)**
Function Description	PWR PVD interrupt callback.
Return values	• None

37.3 PWREx Firmware driver defines

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

37.3.1 PWREx

PWREx

PWR Extended Exported Constants

PWR_EXTI_LINE_PVD External interrupt line 16 Connected to the PVD EXTI Line

PWR Extended Exported Macros

__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_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

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_CLEAR_EGDE_TRIGGER	**Description:** • PVD EXTI line configuration: clear falling edge and rising edge trigger. **Return value:** • None.:
__HAL_PWR_PVD_EXTI_SET_FALLING_EGDE_TRIGGER	**Description:** • PVD EXTI line configuration: set falling edge trigger. **Return value:** • None.:
__HAL_PWR_PVD_EXTI_SET_RISING_EDGE_TRIGGER	**Description:** • PVD EXTI line configuration: set rising edge 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.:

PWR Extended Private Constants

PVD_MODE_IT

PVD_MODE_EVT

PVD_RISING_EDGE

PVD_FALLING_EDGE

PWR Extended 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

IS_PWR_PVD_LEVEL

PWR Extended 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

IS_PWR_PVD_MODE

38 HAL RCC Generic Driver

38.1 RCC Firmware driver registers structures

38.1.1 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32f3xx_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***

38.2 RCC Firmware driver API description

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

38.2.1 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 disabled, 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 which clocks are not derived from the System clock (RTC, ADC, I2C, I2S, TIM, USB FS)

38.2.2 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. The HSI clock can be used also to clock the USART and I2C peripherals.
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 32 MHz 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 different output clocks:
   - The first output is used to generate the high speed system clock (up to 72 MHz)
   - The second output is used to generate the clock for the USB FS (48 MHz)
   - The third output may be used to generate the clock for the ADC peripherals (up to 72 MHz)
   - The fourth output may be used to generate the clock for the TIM peripherals (144 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-M4 NMI (Non-Maskable Interrupt) exception vector.
7. MCO (microcontroller clock output), used to output SYSCLK, HSI, HSE, LSI, LSE or PLL clock (divided by 2) output on pin (such as PA8 pin).

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.
2. All the peripheral clocks are derived from the System clock (SYSCLK) except:
   - The FLASH program/erase clock which is always HSI 8MHz clock.
   - The USB 48 MHz clock which is derived from the PLL VCO clock.
   - The USART clock which can be derived as well from HSI 8MHz, LSI or LSE.
   - The I2C clock which can be derived as well from HSI 8MHz clock.
   - The ADC clock which is derived from PLL output.

   - The RTC clock which is derived from the LSE, LSI or 1 MHz HSE_RTC (HSE divided by a programmable prescaler). The System clock (SYSCLK) frequency must be higher or equal to the RTC clock frequency.
   - IWDG clock which is always the LSI clock.
3. For the STM32F3xx devices, the maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 72 MHz, Depending on the SYSCLK frequency, the flash latency should be adapted accordingly (see *Table 25: "Number of wait states (WS) according to system clock (SYSCLK) frequency"*).
4. After reset, the System clock source is the HSI (8 MHz) with 0 WS and prefetch is disabled.

Table 25: Number of wait states (WS) according to system clock (SYSCLK) frequency

Latency	SYSCLK clock frequency (MHz)
0WS (1CPU cycle)	0 ≤ SYSCLK ≤ 24
1WS (2CPU cycles)	24 < HCLK ≤ 48
2 WS (3CPU cycles)	48 < HCLK ≤ 72

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

38.2.3 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_CCSCallback()***

38.2.4 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, MCO OFFAll interrupts disabled • This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

38.2.5 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.

38.2.6 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
Return values	• HAL status
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.

38.2.7 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(such as PA8).

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 MCO pin(such as PA8). • **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values: RCC_MCOSOURCE_LSI: LSI clock selected as MCO source RCC_MCOSOURCE_HSI: HSI clock selected as MCO source RCC_MCOSOURCE_LSE: LSE clock selected as MCO source RCC_MCOSOURCE_HSE: HSE clock selected as MCO source RCC_MCOSOURCE_PLLCLK_DIV2: main PLL clock divided by 2 selected as MCO source RCC_MCOSOURCE_SYSCLK: System clock (SYSCLK) selected as MCO source • **RCC_MCODiv:** specifies the MCOx prescaler. This parameter can be one of the following values: RCC_MCO_NODIV: no division applied to MCO clock
Return values	• None
Notes	• MCO pin (such as PA8) should be configured in alternate function mode.

38.2.8 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-M4 NMI (Non-Maskable Interrupt) exception vector.

38.2.9 HAL_RCC_DisableCSS

Function Name	**void HAL_RCC_DisableCSS (void)**
Function Description	Disables the Clock Security System.
Return values	• None

38.2.10 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 a value based on HSI_VALUE(*) • If SYSCLK source is HSE, function returns a value based on HSE_VALUE divided by PREDIV factor(**) • If SYSCLK source is PLL, function returns a value based on HSE_VALUE divided by PREDIV factor(**) or HSI_VALUE(*) multiplied by the PLL factor. • (*) HSI_VALUE is a constant defined in stm32f3xx.h file (default value 8 MHz). • (**) HSE_VALUE is a constant defined in stm32f3xx.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.

38.2.11 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

38.2.12 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.

38.2.13 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.

38.2.14 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

38.2.15 HAL_RCC_GetClockConfig

Function Name	**void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)**
Function Description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	• **RCC_ClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that contains the current clock configuration. • **pFLatency:** Pointer on the Flash Latency.
Return values	• None

38.2.16 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().

38.2.17 HAL_RCC_CCSCallback

Function Name	**void HAL_RCC_CCSCallback (void)**
Function Description	RCC Clock Security System interrupt callback.
Return values	• None

38.3 RCC Firmware driver defines

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

38.3.1 RCC

RCC

RCC AHB Clock Enable Disable

__GPIOA_CLK_ENABLE

__GPIOB_CLK_ENABLE

__GPIOC_CLK_ENABLE

__GPIOD_CLK_ENABLE

__GPIOF_CLK_ENABLE

__CRC_CLK_ENABLE

__DMA1_CLK_ENABLE

__SRAM_CLK_ENABLE

__FLITF_CLK_ENABLE

__TSC_CLK_ENABLE

__GPIOA_CLK_DISABLE

__GPIOB_CLK_DISABLE

__GPIOC_CLK_DISABLE

__GPIOD_CLK_DISABLE

__GPIOF_CLK_DISABLE

__CRC_CLK_DISABLE

__DMA1_CLK_DISABLE

__SRAM_CLK_DISABLE

__FLITF_CLK_DISABLE

__TSC_CLK_DISABLE

RCC AHB Clock Source

RCC_SYSCLK_DIV1

RCC_SYSCLK_DIV2

RCC_SYSCLK_DIV4

RCC_SYSCLK_DIV8

RCC_SYSCLK_DIV16

RCC_SYSCLK_DIV64

RCC_SYSCLK_DIV128

RCC_SYSCLK_DIV256

RCC_SYSCLK_DIV512

IS_RCC_SYSCLK_DIV

RCC AHB Force Release Reset

__AHB_FORCE_RESET

__GPIOA_FORCE_RESET

__GPIOB_FORCE_RESET

__GPIOC_FORCE_RESET

__GPIOD_FORCE_RESET

__GPIOF_FORCE_RESET

__TSC_FORCE_RESET

__AHB_RELEASE_RESET

__GPIOA_RELEASE_RESET

__GPIOB_RELEASE_RESET

__GPIOC_RELEASE_RESET

__GPIOD_RELEASE_RESET

__GPIOF_RELEASE_RESET

__TSC_RELEASE_RESET

RCC APB1 APB2 Clock Source

RCC_HCLK_DIV1

RCC_HCLK_DIV2

RCC_HCLK_DIV4

RCC_HCLK_DIV8

RCC_HCLK_DIV16

IS_RCC_HCLK_DIV

RCC APB1 Clock Enable Disable

__TIM2_CLK_ENABLE

__TIM6_CLK_ENABLE

__WWDG_CLK_ENABLE

__USART2_CLK_ENABLE

__USART3_CLK_ENABLE

__I2C1_CLK_ENABLE

__PWR_CLK_ENABLE

__DAC1_CLK_ENABLE

__TIM2_CLK_DISABLE

__TIM6_CLK_DISABLE

__WWDG_CLK_DISABLE

__USART2_CLK_DISABLE

__USART3_CLK_DISABLE

__I2C1_CLK_DISABLE

__PWR_CLK_DISABLE

__DAC1_CLK_DISABLE

RCC APB1 Force Release Reset

__APB1_FORCE_RESET

__TIM2_FORCE_RESET

__TIM6_FORCE_RESET

__WWDG_FORCE_RESET

__USART2_FORCE_RESET

__USART3_FORCE_RESET

__I2C1_FORCE_RESET

__PWR_FORCE_RESET

__DAC1_FORCE_RESET

__APB1_RELEASE_RESET

__TIM2_RELEASE_RESET

__TIM6_RELEASE_RESET

__WWDG_RELEASE_RESET

__USART2_RELEASE_RESET

__USART3_RELEASE_RESET

__I2C1_RELEASE_RESET

__PWR_RELEASE_RESET

__DAC1_RELEASE_RESET

RCC APB2 Clock Enable Disable

__SYSCFG_CLK_ENABLE

__TIM15_CLK_ENABLE

__TIM16_CLK_ENABLE

__TIM17_CLK_ENABLE

__USART1_CLK_ENABLE

__SYSCFG_CLK_DISABLE

__TIM15_CLK_DISABLE

__TIM16_CLK_DISABLE

__TIM17_CLK_DISABLE

__USART1_CLK_DISABLE

RCC APB2 Force Release Reset

__APB2_FORCE_RESET

__SYSCFG_FORCE_RESET

__TIM15_FORCE_RESET

__TIM16_FORCE_RESET

__TIM17_FORCE_RESET

__USART1_FORCE_RESET

__APB2_RELEASE_RESET

__SYSCFG_RELEASE_RESET

__TIM15_RELEASE_RESET

__TIM16_RELEASE_RESET

__TIM17_RELEASE_RESET

__USART1_RELEASE_RESET

RCC BitAddress AliasRegion

RCC_OFFSET

RCC_CR_OFFSET

HSION_BitNumber

CR_HSION_BB

HSEON_BitNumber

CR_HSEON_BB

CSSON_BitNumber

CR_CSSON_BB

PLLON_BitNumber

CR_PLLON_BB

RCC_CFGR_OFFSET

PLLSRC_BitNumber

CFGR_PLLSRC_BB

RCC_CIR_OFFSET

RCC_BDCR_OFFSET

LSEON_BitNumber

BDCR_LSEON_BB

RTCEN_BitNumber

BDCR_RTCEN_BB

BDRST_BitNumber

BDCR_BDRST_BB

RCC_CSR_OFFSET

LSION_BitNumber

CSR_LSION_BB

RMVF_BitNumber

CSR_RMVF_BB

CR_BYTE2_ADDRESS

CIR_BYTE1_ADDRESS

CIR_BYTE2_ADDRESS

CSR_BYTE1_ADDRESS

BDCR_BYTE0_ADDRESS

RCC Flag

CR_REG_INDEX

BDCR_REG_INDEX

CSR_REG_INDEX

RCC_FLAG_HSIRDY

RCC_FLAG_HSERDY

RCC_FLAG_PLLRDY

RCC_FLAG_LSERDY

RCC_FLAG_LSIRDY

RCC_FLAG_RMV

RCC_FLAG_OBLRST

RCC_FLAG_PINRST

RCC_FLAG_PORRST

RCC_FLAG_SFTRST

RCC_FLAG_IWDGRST

RCC_FLAG_WWDGRST

RCC_FLAG_LPWRRST

RCC Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:

- Enable RCC interrupt (Perform Byte access to RCC_CIR[12:8] 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 enable
  - RCC_IT_LSERDY: LSE ready interrupt enable
  - RCC_IT_HSIRDY: HSI ready interrupt enable
  - RCC_IT_HSERDY: HSE ready interrupt enable

- RCC_IT_PLLRDY: PLL ready interrupt enable

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt (Perform Byte access to RCC_CIR[12:8] 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_LSIRDYIE: LSI ready interrupt enable
  - RCC_IT_LSERDYIE: LSE ready interrupt enable
  - RCC_IT_HSIRDYIE: HSI ready interrupt enable
  - RCC_IT_HSERDYIE: HSE ready interrupt enable
  - RCC_IT_PLLRDYIE: PLL ready interrupt enable

__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:

- __IT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDYC: LSI ready interrupt clear
  - RCC_IT_LSERDYC: LSE ready interrupt clear
  - RCC_IT_HSIRDYC: HSI ready interrupt clear
  - RCC_IT_HSERDYC: HSE ready interrupt clear
  - RCC_IT_PLLRDYC: PLL ready interrupt clear
  - RCC_IT_CSSC: Clock Security System interrupt clear

__HAL_RCC_GET_IT

Description:

- Check the RCC's interrupt has occurred or not.

Parameters:

- __IT__: specifies the RCC interrupt source to check. This parameter can be one of the

following values:
- RCC_IT_LSIRDYF: LSI ready interrupt flag
- RCC_IT_LSERDYF: LSE ready interrupt flag
- RCC_IT_HSIRDYF: HSI ready interrupt flag
- RCC_IT_HSERDYF: HSE ready interrupt flag
- RCC_IT_PLLRDYF: PLL ready interrupt flag
- RCC_IT_CSSF: Clock Security System interrupt flag

Return value:

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

__HAL_RCC_CLEAR_RESET_FLAGS

RCC_FLAG_MASK

Description:

- Check RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_FLAG_HSIRDY: HSI oscillator clock ready
  - RCC_FLAG_HSERDY: HSE oscillator clock ready
  - RCC_FLAG_PLLRDY: PLL clock ready
  - RCC_FLAG_LSERDY: LSE oscillator clock ready
  - RCC_FLAG_LSIRDY: LSI oscillator clock ready
  - RCC_FLAG_OBLRST: Option Byte Load reset
  - 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).

__HAL_RCC_GET_FLAG

RCC Force Release Backup

__HAL_RCC_BACKUPRESET_FORCE

__HAL_RCC_BACKUPRESET_RELEASE

RCC 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 value:
  - 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:

- Macro to get the oscillator used as PLL clock source.

Return value:

- The: oscillator used as PLL clock source. The returned value can be one of the following:
  - RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

RCC HSE Config

RCC_HSE_OFF

RCC_HSE_ON

RCC_HSE_BYPASS

IS_RCC_HSE

RCC 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

RCC HSI Config

RCC_HSI_OFF

RCC_HSI_ON

IS_RCC_HSI

RCC_HSICALIBRATION_DEFAULT

IS_RCC_CALIBRATION_VALUE

RCC 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. This parameter must be a number between 0 and 0x1F.

RCC I2C1 Clock Source

RCC_I2C1CLKSOURCE_HSI

RCC_I2C1CLKSOURCE_SYSCLK

IS_RCC_I2C1CLKSOURCE

RCC I2Cx Clock Config

__HAL_RCC_I2C1_CONFIG

Description:

- Macro to configure the I2C1 clock (I2C1CLK).

Parameters:

- __I2C1CLKSource__: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC_I2C1CLKSOURCE_HSI: HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK: System Clock selected as I2C1 clock

__HAL_RCC_GET_I2C1_SOURCE

Description:

- Macro to get the I2C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C1CLKSOURCE_HSI: HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK:

System Clock selected as I2C1 clock

RCC Interrupt

RCC_IT_LSIRDY

RCC_IT_LSERDY

RCC_IT_HSIRDY

RCC_IT_HSERDY

RCC_IT_PLLRDY

RCC_IT_CSS

RCC_LSE_Config

RCC_LSE_OFF

RCC_LSE_ON

RCC_LSE_BYPASS

IS_RCC_LSE

RCC LSE Configuration

__HAL_RCC_LSE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- __STATE__: specifies the new state of the LSE. This parameter can be one of the following values:
  – RCC_LSE_OFF: turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  – RCC_LSE_ON: turn ON the LSE oscillator
  – RCC_LSE_BYPASS: LSE oscillator bypassed with external clock

RCC LSI Config

RCC_LSI_OFF

RCC_LSI_ON

IS_RCC_LSI

RCC LSI Configuration

__HAL_RCC_LSI_ENABLE

__HAL_RCC_LSI_DISABLE

RCC MCOx Index

RCC_MCO

IS_RCC_MCO

RCC Oscillator Type

RCC_OSCILLATORTYPE_NONE

RCC_OSCILLATORTYPE_HSE

RCC_OSCILLATORTYPE_HSI

RCC_OSCILLATORTYPE_LSE

RCC_OSCILLATORTYPE_LSI

IS_RCC_OSCILLATORTYPE

RCC PLL Config

RCC_PLL_NONE

RCC_PLL_OFF

RCC_PLL_ON

IS_RCC_PLL

RCC PLL Configuration

__HAL_RCC_PLL_ENABLE

__HAL_RCC_PLL_DISABLE

RCC 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

IS_RCC_PLL_MUL

RCC Private Define

HSE_TIMEOUT_VALUE

HSI_TIMEOUT_VALUE

LSI_TIMEOUT_VALUE

PLL_TIMEOUT_VALUE

CLOCKSWITCH_TIMEOUT_VALUE

RCC Private Macros

__MCO_CLK_ENABLE

MCO_GPIO_PORT

MCO_PIN

RCC RTC Clock Configuration

__HAL_RCC_RTC_ENABLE

__HAL_RCC_RTC_DISABLE

__HAL_RCC_RTC_CONFIG

Description:

- Macro to configure the RTC clock (RTCCLK).

Parameters:

- __RTCCLKSource__: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC_RTCCLKSOURCE_NONE: No clock selected as RTC clock
  - RCC_RTCCLKSOURCE_LSE: LSE selected as RTC clock
  - RCC_RTCCLKSOURCE_LSI: LSI selected as RTC clock
  - RCC_RTCCLKSOURCE_HSE_DIV32: HSE clock divided by 32

__HAL_RCC_GET_RTC_SOURCE

Description:

- Macro to get the RTC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_RTCCLKSOURCE_NONE: No clock selected as RTC clock
  - RCC_RTCCLKSOURCE_LSE: LSE selected as RTC clock
  - RCC_RTCCLKSOURCE_LSI: LSI selected as RTC clock
  - RCC_RTCCLKSOURCE_HSE_DIV32: HSE clock divided by 32 selected as RTC clock

RCC RTC Clock Source

RCC_RTCCLKSOURCE_NONE

RCC_RTCCLKSOURCE_LSE

RCC_RTCCLKSOURCE_LSI

RCC_RTCCLKSOURCE_HSE_DIV32

IS_RCC_RTCCLKSOURCE

RCC System Clock Source

RCC_SYSCLKSOURCE_HSI

RCC_SYSCLKSOURCE_HSE

RCC_SYSCLKSOURCE_PLLCLK

IS_RCC_SYSCLKSOURCE

RCC System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_HSI

RCC_SYSCLKSOURCE_STATUS_HSE

RCC_SYSCLKSOURCE_STATUS_PLLCLK

IS_RCC_SYSCLKSOURCE_STATUS

RCC System Clock Type

RCC_CLOCKTYPE_SYSCLK

RCC_CLOCKTYPE_HCLK

RCC_CLOCKTYPE_PCLK1

RCC_CLOCKTYPE_PCLK2

IS_RCC_CLOCKTYPE

RCC Timeout

LSE_TIMEOUT_VALUE

DBP_TIMEOUT_VALUE

RCC USART2 Clock Source

RCC_USART2CLKSOURCE_PCLK1

RCC_USART2CLKSOURCE_SYSCLK

RCC_USART2CLKSOURCE_LSE

RCC_USART2CLKSOURCE_HSI

IS_RCC_USART2CLKSOURCE

RCC USART3 Clock Source

RCC_USART3CLKSOURCE_PCLK1

RCC_USART3CLKSOURCE_SYSCLK

RCC_USART3CLKSOURCE_LSE

RCC_USART3CLKSOURCE_HSI

IS_RCC_USART3CLKSOURCE

RCC USARTx Clock Config

__HAL_RCC_USART1_CONFIG

Description:

- Macro to configure the USART1 clock (USART1CLK).

Parameters:

- __USART1CLKSource__: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 or RCC_USART1CLKSOURCE_PCLK1: PCLK2 or PCLK1 selected as USART1

clock
- RCC_USART1CLKSOURCE_HSI: HSI selected as USART1 clock
- RCC_USART1CLKSOURCE_SYSCLK: System Clock selected as USART1 clock
- RCC_USART1CLKSOURCE_LSE: LSE selected as USART1 clock

__HAL_RCC_GET_USART1_SOURCE

Description:

- Macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 or RCC_USART1CLKSOURCE_PCLK1: PCLK2 or PCLK1 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI: HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK: System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE: LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- __USART2CLKSource__: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1: PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI: HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK: System Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE: LSE selected as USART2 clock

__HAL_RCC_GET_USART2_SOURCE

Description:

- Macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1: PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI: HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK: System Clock selected as USART2 clock

- RCC_USART2CLKSOURCE_LSE: LSE selected as USART2 clock

__HAL_RCC_USART3_CONFIG

Description:

- Macro to configure the USART3 clock (USART3CLK).

Parameters:

- __USART3CLKSource__: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1: PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI: HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_SYSCLK: System Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE: LSE selected as USART3 clock

__HAL_RCC_GET_USART3_SOURCE

Description:

- Macro to get the USART3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1: PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI: HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_SYSCLK: System Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE: LSE selected as USART3 clock

39 HAL RCC Extension Driver

39.1 RCCEx Firmware driver registers structures

39.1.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the stm32f3xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLMUL***
- ***uint32_t PREDIV***

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***
  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 ***RCCEx_PLL_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLMUL***
  PLLMUL: Multiplication factor for PLL VCO input clock This parameter must be a value of ***RCC_PLL_Multiplication_Factor***
- ***uint32_t RCC_PLLInitTypeDef::PREDIV***
  PREDIV: Predivision factor for PLL VCO input clock This parameter must be a value of ***RCCEx_PLL_Prediv_Factor***

39.1.2 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32f3xx_hal_rcc_ex.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***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::LSEState***
  The new state of the LSE. This parameter can be a value of ***RCC_LSE_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIState***
  The new state of the HSI. This parameter can be a value of ***RCC_HSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
  The calibration trimming value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 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

39.1.3 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32f3xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PeriphClockSelection***
- ***uint32_t RTCClockSelection***
- ***uint32_t Usart1ClockSelection***
- ***uint32_t Usart2ClockSelection***
- ***uint32_t Usart3ClockSelection***
- ***uint32_t Uart4ClockSelection***
- ***uint32_t Uart5ClockSelection***
- ***uint32_t I2c1ClockSelection***
- ***uint32_t I2c2ClockSelection***
- ***uint32_t I2c3ClockSelection***
- ***uint32_t Adc12ClockSelection***
- ***uint32_t Adc34ClockSelection***
- ***uint32_t I2sClockSelection***
- ***uint32_t Tim1ClockSelection***
- ***uint32_t Tim2ClockSelection***
- ***uint32_t Tim34ClockSelection***
- ***uint32_t Tim8ClockSelection***
- ***uint32_t Tim15ClockSelection***
- ***uint32_t Tim16ClockSelection***
- ***uint32_t Tim17ClockSelection***
- ***uint32_t Tim20ClockSelection***
- ***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 RTC Clock Prescalers Selection This parameter can be a value of ***RCC_RTC_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection***
  USART1 clock source This parameter can be a value of ***RCCEx_USART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection***
  USART2 clock source This parameter can be a value of ***RCC_USART2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart3ClockSelection***
  USART3 clock source This parameter can be a value of ***RCC_USART3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart4ClockSelection***
  UART4 clock source This parameter can be a value of ***RCCEx_UART4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart5ClockSelection***
  UART5 clock source This parameter can be a value of ***RCCEx_UART5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection***
  I2C1 clock source This parameter can be a value of ***RCC_I2C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c2ClockSelection***
  I2C2 clock source This parameter can be a value of ***RCCEx_I2C2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection***
  I2C3 clock source This parameter can be a value of ***RCCEx_I2C3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Adc12ClockSelection***
  ADC1 & ADC2 clock source This parameter can be a value of ***RCCEx_ADC12_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Adc34ClockSelection***
  ADC3 & ADC4 clock source This parameter can be a value of ***RCCEx_ADC34_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2sClockSelection***
  I2S clock source This parameter can be a value of ***RCCEx_I2S_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Tim1ClockSelection***
  TIM1 clock source This parameter can be a value of ***RCCEx_TIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Tim2ClockSelection***
  TIM2 clock source This parameter can be a value of ***RCCEx_TIM2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Tim34ClockSelection***
  TIM3 & TIM4 clock source This parameter can be a value of ***RCCEx_TIM34_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Tim8ClockSelection***
  TIM8 clock source This parameter can be a value of ***RCCEx_TIM8_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Tim15ClockSelection***
  TIM15 clock source This parameter can be a value of ***RCCEx_TIM15_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Tim16ClockSelection***
  TIM16 clock source This parameter can be a value of ***RCCEx_TIM16_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Tim17ClockSelection***
  TIM17 clock source This parameter can be a value of ***RCCEx_TIM17_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Tim20ClockSelection***
  TIM20 clock source This parameter can be a value of ***RCCEx_TIM20_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::USBClockSelection***
  USB clock source This parameter can be a value of ***RCCEx_USB_Clock_Source***

39.2 RCCEx Firmware driver API description

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

39.2.1 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_RCC_OscConfig()*
- *HAL_RCC_GetOscConfig()*
- *HAL_RCC_GetSysClockFreq()*

39.2.2 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 (ADC, CEC, I2C, I2S, SDADC, HRTIM, TIM, USART, RTC and USB).
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) and RCC_BDCR register are set to their reset values.

39.2.3 HAL_RCCEx_GetPeriphCLKConfig

Function Name	**void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)**
Function Description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	• **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks (ADC, CEC, I2C, I2S, SDADC, HRTIM, TIM, USART, RTC and USB clocks).
Return values	• None

39.2.4 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.

39.2.5 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

39.2.6 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 a value based on HSI_VALUE(*) • If SYSCLK source is HSE, function returns a value based on HSE_VALUE divided by PREDIV factor(**) • If SYSCLK source is PLL, function returns a value based on HSE_VALUE divided by PREDIV factor(**) or HSI_VALUE(*) multiplied by the PLL factor. • (*) HSI_VALUE is a constant defined in stm32f3xx.h file (default value 8 MHz). • (**) HSE_VALUE is a constant defined in stm32f3xx.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.

39.3 RCCEx Firmware driver defines

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

39.3.1 RCCEx

RCCEx

RCC Extended ADC12 Clock Source

RCC_ADC12PLLCLK_OFF

RCC_ADC12PLLCLK_DIV1

RCC_ADC12PLLCLK_DIV2

RCC_ADC12PLLCLK_DIV4

RCC_ADC12PLLCLK_DIV6

RCC_ADC12PLLCLK_DIV8

RCC_ADC12PLLCLK_DIV10

RCC_ADC12PLLCLK_DIV12

RCC_ADC12PLLCLK_DIV16

RCC_ADC12PLLCLK_DIV32

RCC_ADC12PLLCLK_DIV64

RCC_ADC12PLLCLK_DIV128

RCC_ADC12PLLCLK_DIV256

IS_RCC_ADC12PLLCLK_DIV

RCC Extended ADC34 Clock Source

RCC_ADC34PLLCLK_OFF

RCC_ADC34PLLCLK_DIV1

RCC_ADC34PLLCLK_DIV2

RCC_ADC34PLLCLK_DIV4

RCC_ADC34PLLCLK_DIV6

RCC_ADC34PLLCLK_DIV8

RCC_ADC34PLLCLK_DIV10

RCC_ADC34PLLCLK_DIV12

RCC_ADC34PLLCLK_DIV16

RCC_ADC34PLLCLK_DIV32

RCC_ADC34PLLCLK_DIV64

RCC_ADC34PLLCLK_DIV128

RCC_ADC34PLLCLK_DIV256

IS_RCC_ADC34PLLCLK_DIV

RCC Extended ADCx Clock Config

__HAL_RCC_ADC12_CONFIG

Description:

- Macro to configure the ADC1 & ADC2 clock (ADC12CLK).

Parameters:

- __ADC12CLKSource__: specifies the ADC1 & ADC2 clock source. This parameter can be one of the following values:
  - RCC_ADC12PLLCLK_OFF: ADC1 & ADC2 PLL clock disabled, ADC1 & ADC2 can use AHB clock
  - RCC_ADC12PLLCLK_DIV1: PLL clock divided by 1 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV2: PLL clock divided by 2 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV4: PLL clock divided by 4 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV6: PLL clock divided by 6 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV8: PLL clock divided by 8 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV10: PLL clock divided by 10 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV12: PLL clock divided by 12 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV16: PLL clock divided by 16 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV32: PLL clock divided by 32 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV64: PLL clock divided by 64 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV128: PLL clock divided by 128 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV256: PLL clock divided by 256 selected as ADC1 & ADC2 clock

__HAL_RCC_GET_ADC12_SOURCE

Description:

- Macro to get the ADC1 & ADC2 clock.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADC12PLLCLK_OFF: ADC1 & ADC2 PLL clock disabled, ADC1 & ADC2 can use AHB clock
  - RCC_ADC12PLLCLK_DIV1: PLL clock divided by 1 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV2: PLL clock divided by 2 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV4: PLL clock divided by 4 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV6: PLL clock divided by 6 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV8: PLL clock divided by 8 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV10: PLL clock divided by 10 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV12: PLL clock divided by 12 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV16: PLL clock divided by 16 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV32: PLL clock divided by 32 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV64: PLL clock divided by 64 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV128: PLL clock divided by 128 selected as ADC1 & ADC2 clock
  - RCC_ADC12PLLCLK_DIV256: PLL clock divided by 256 selected as ADC1 & ADC2 clock

__HAL_RCC_ADC34_CONFIG

Description:

- Macro to configure the ADC3 & ADC4 clock (ADC34CLK).

Parameters:

- __ADC34CLKSource__: specifies the ADC3 & ADC4 clock source. This parameter can be one of the following values:

- RCC_ADC34PLLCLK_OFF: ADC3 & ADC4 PLL clock disabled, ADC3 & ADC4 can use AHB clock
- RCC_ADC34PLLCLK_DIV1: PLL clock divided by 1 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV2: PLL clock divided by 2 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV4: PLL clock divided by 4 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV6: PLL clock divided by 6 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV8: PLL clock divided by 8 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV10: PLL clock divided by 10 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV12: PLL clock divided by 12 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV16: PLL clock divided by 16 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV32: PLL clock divided by 32 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV64: PLL clock divided by 64 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV128: PLL clock divided by 128 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV256: PLL clock divided by 256 selected as ADC3 & ADC4 clock

__HAL_RCC_GET_ADC34_SOURCE

Description:

- Macro to get the ADC3 & ADC4 clock.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADC34PLLCLK_OFF: ADC3 & ADC4 PLL clock disabled, ADC3 & ADC4 can use AHB clock
  - RCC_ADC34PLLCLK_DIV1: PLL clock divided by 1 selected as ADC3 & ADC4 clock
  - RCC_ADC34PLLCLK_DIV2: PLL clock divided by 2 selected as ADC3 & ADC4

clock

- RCC_ADC34PLLCLK_DIV4: PLL clock divided by 4 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV6: PLL clock divided by 6 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV8: PLL clock divided by 8 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV10: PLL clock divided by 10 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV12: PLL clock divided by 12 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV16: PLL clock divided by 16 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV32: PLL clock divided by 32 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV64: PLL clock divided by 64 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV128: PLL clock divided by 128 selected as ADC3 & ADC4 clock
- RCC_ADC34PLLCLK_DIV256: PLL clock divided by 256 selected as ADC3 & ADC4 clock

RCC Extended AHB Clock Enable Disable

__DMA2_CLK_ENABLE

__GPIOE_CLK_ENABLE

__ADC12_CLK_ENABLE

__ADC1_CLK_ENABLE

__ADC2_CLK_ENABLE

__DMA2_CLK_DISABLE

__GPIOE_CLK_DISABLE

__ADC12_CLK_DISABLE

__ADC1_CLK_DISABLE

__ADC2_CLK_DISABLE

__ADC34_CLK_ENABLE

__ADC34_CLK_DISABLE

__FMC_CLK_ENABLE

__GPIOG_CLK_ENABLE

__GPIOH_CLK_ENABLE

__FMC_CLK_DISABLE

__GPIOG_CLK_DISABLE

__GPIOH_CLK_DISABLE

RCC Extended AHB Force Release Reset

__GPIOE_FORCE_RESET

__ADC12_FORCE_RESET

__ADC1_FORCE_RESET

__ADC2_FORCE_RESET

__GPIOE_RELEASE_RESET

__ADC12_RELEASE_RESET

__ADC1_RELEASE_RESET

__ADC2_RELEASE_RESET

__ADC34_FORCE_RESET

__ADC34_RELEASE_RESET

__FMC_FORCE_RESET

__GPIOG_FORCE_RESET

__GPIOH_FORCE_RESET

__FMC_RELEASE_RESET

__GPIOG_RELEASE_RESET

__GPIOH_RELEASE_RESET

RCC Extended APB1 Clock Enable Disable

__TIM3_CLK_ENABLE

__TIM4_CLK_ENABLE

__SPI2_CLK_ENABLE

__SPI3_CLK_ENABLE

__UART4_CLK_ENABLE

__UART5_CLK_ENABLE

__I2C2_CLK_ENABLE

__TIM3_CLK_DISABLE

__TIM4_CLK_DISABLE

__SPI2_CLK_DISABLE

__SPI3_CLK_DISABLE

__UART4_CLK_DISABLE

__UART5_CLK_DISABLE

__I2C2_CLK_DISABLE

__TIM7_CLK_ENABLE

__TIM7_CLK_DISABLE

__USB_CLK_ENABLE

__USB_CLK_DISABLE

__CAN_CLK_ENABLE

__CAN_CLK_DISABLE

__I2C3_CLK_ENABLE

__I2C3_CLK_DISABLE

RCC Extended APB1 Force Release Reset

__TIM3_FORCE_RESET

__TIM4_FORCE_RESET

__SPI2_FORCE_RESET

__SPI3_FORCE_RESET

__UART4_FORCE_RESET

__UART5_FORCE_RESET

__I2C2_FORCE_RESET

__TIM3_RELEASE_RESET

__TIM4_RELEASE_RESET

__SPI2_RELEASE_RESET

__SPI3_RELEASE_RESET

__UART4_RELEASE_RESET

__UART5_RELEASE_RESET

__I2C2_RELEASE_RESET

__TIM7_FORCE_RESET

__TIM7_RELEASE_RESET

__USB_FORCE_RESET

__USB_RELEASE_RESET

__CAN_FORCE_RESET

__CAN_RELEASE_RESET

__I2C3_FORCE_RESET

__I2C3_RELEASE_RESET

RCC Extended APB2 Clock Enable Disable

__SPI1_CLK_ENABLE

__SPI1_CLK_DISABLE

__TIM8_CLK_ENABLE

__TIM8_CLK_DISABLE

__TIM1_CLK_ENABLE

__TIM1_CLK_DISABLE

__SPI4_CLK_ENABLE

__SPI4_CLK_DISABLE

__TIM20_CLK_ENABLE

__TIM20_CLK_DISABLE

RCC Extended APB2 Force Release Reset

__SPI1_FORCE_RESET

__SPI1_RELEASE_RESET

__TIM8_FORCE_RESET

__TIM8_RELEASE_RESET

__TIM1_FORCE_RESET

__TIM1_RELEASE_RESET

__SPI4_FORCE_RESET

__SPI4_RELEASE_RESET

__TIM20_FORCE_RESET

__TIM20_RELEASE_RESET

RCC Extended I2C2 Clock Source

RCC_I2C2CLKSOURCE_HSI

RCC_I2C2CLKSOURCE_SYSCLK

IS_RCC_I2C2CLKSOURCE

RCC Extended I2C3 Clock Source

RCC_I2C3CLKSOURCE_HSI

RCC_I2C3CLKSOURCE_SYSCLK

IS_RCC_I2C3CLKSOURCE

RCC Extended I2Cx Clock Config

__HAL_RCC_I2C2_CONFIG

Description:

- Macro to configure the I2C2 clock (I2C2CLK).

Parameters:

- __I2C2CLKSource__: specifies the I2C2 clock source. This parameter can be one of the following values:
  - RCC_I2C2CLKSOURCE_HSI: HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_SYSCLK: System Clock selected as I2C2 clock

__HAL_RCC_GET_I2C2_SOURCE

Description:

- Macro to get the I2C2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C2CLKSOURCE_HSI: HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_SYSCLK: System Clock selected as I2C2 clock

__HAL_RCC_I2C3_CONFIG

Description:

- Macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- __I2C3CLKSource__: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC_I2C3CLKSOURCE_HSI: HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_SYSCLK: System Clock selected as I2C3 clock

__HAL_RCC_GET_I2C3_SOURCE

Description:

- Macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C3CLKSOURCE_HSI: HSI selected as I2C3 clock
  - RCC_I2C3CLKSOURCE_SYSCLK: System Clock selected as I2C3 clock

RCC Extended I2Sx Clock Config

__HAL_RCC_I2S_CONFIG

Description:

- Macro to configure the I2S clock source (I2SCLK).

Parameters:

- __I2SCLKSource__: specifies the I2S clock source. This parameter can be one of the following values:
  - RCC_I2SCLKSOURCE_SYSCLK: SYSCLK clock used as I2S clock source
  - RCC_I2SCLKSOURCE_EXT: External clock mapped on the I2S_CKIN pin used as I2S clock source

__HAL_RCC_GET_I2S_SOURCE

Description:

- Macro to get the I2S clock source (I2SCLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_I2SCLKSOURCE_SYSCLK: SYSCLK

clock used as I2S clock source
  - RCC_I2SCLKSOURCE_EXT: External clock mapped on the I2S_CKIN pin used as I2S clock source

RCC Extended I2S Clock Source

RCC_I2SCLKSOURCE_SYSCLK

RCC_I2SCLKSOURCE_EXT

IS_RCC_I2SCLKSOURCE

RCC Extended MCOx Clock Config

__HAL_RCC_MCO_CONFIG

Description:

- macro to configure the MCO clock.

Parameters:

- __MCOCLKSource__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCOSOURCE_HSI: HSI selected as MCO clock
  - RCC_MCOSOURCE_HSE: HSE selected as MCO clock
  - RCC_MCOSOURCE_LSI: LSI selected as MCO clock
  - RCC_MCOSOURCE_LSE: LSE selected as MCO clock
  - RCC_MCOSOURCE_PLLCLK_DIV2: PLLCLK Divided by 2 selected as MCO clock
  - RCC_MCOSOURCE_SYSCLK: System Clock selected as MCO clock
- __MCODiv__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCO_NODIV: No division applied on MCO clock source

RCC Extended MCOx Clock Prescaler

RCC_MCO_DIV1

RCC_MCO_DIV2

RCC_MCO_DIV4

RCC_MCO_DIV8

RCC_MCO_DIV16

RCC_MCO_DIV32

RCC_MCO_DIV64

RCC_MCO_DIV128

IS_RCC_MCODIV

RCC Extended MCO Clock Source

RCC_MCOSOURCE_NONE

RCC_MCOSOURCE_LSI

RCC_MCOSOURCE_LSE

RCC_MCOSOURCE_SYSCLK

RCC_MCOSOURCE_HSI

RCC_MCOSOURCE_HSE

RCC_MCOSOURCE_PLLCLK_DIV1

RCC_MCOSOURCE_PLLCLK_DIV2

IS_RCC_MCOSOURCE

RCC Extended Periph Clock Selection

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_USART3

RCC_PERIPHCLK_UART4

RCC_PERIPHCLK_UART5

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_ADC12

RCC_PERIPHCLK_ADC34

RCC_PERIPHCLK_I2S

RCC_PERIPHCLK_TIM1

RCC_PERIPHCLK_TIM8

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_USB

RCC_PERIPHCLK_I2C3

RCC_PERIPHCLK_TIM2

RCC_PERIPHCLK_TIM34

RCC_PERIPHCLK_TIM15

RCC_PERIPHCLK_TIM16

RCC_PERIPHCLK_TIM17

RCC_PERIPHCLK_TIM20

IS_RCC_PERIPHCLK

RCC Extended PLL Clock Source

RCC_PLLSOURCE_HSI

RCC_PLLSOURCE_HSE

IS_RCC_PLLSOURCE

RCC Extended PLL Configuration

__HAL_RCC_PLL_CONFIG	**Description:** • Macro to configure the PLL clock source, multiplication and division factors. **Parameters:** • __RCC_PLLSource__: specifies the PLL entry clock source. This parameter can be one of the following values: – RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL clock entry – RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL clock entry • __PREDIV__: specifies the predivider factor for PLL VCO input clock This parameter must be a number between RCC_PREDIV_DIV1 and RCC_PREDIV_DIV16. • __PLLMUL__: specifies the multiplication factor for PLL VCO input clock This parameter must be a number between RCC_PLL_MUL2 and RCC_PLL_MUL16.

RCC Extended PLL Prediv Factor

RCC_PREDIV_DIV1

RCC_PREDIV_DIV2

RCC_PREDIV_DIV3

RCC_PREDIV_DIV4

RCC_PREDIV_DIV5

RCC_PREDIV_DIV6

RCC_PREDIV_DIV7

RCC_PREDIV_DIV8

RCC_PREDIV_DIV9

RCC_PREDIV_DIV10

RCC_PREDIV_DIV11

RCC_PREDIV_DIV12

RCC_PREDIV_DIV13

RCC_PREDIV_DIV14

RCC_PREDIV_DIV15

RCC_PREDIV_DIV16

IS_RCC_PREDIV

RCC Extended Private Define

HSE_TIMEOUT_VALUE

HSI_TIMEOUT_VALUE

LSI_TIMEOUT_VALUE

PLL_TIMEOUT_VALUE

CLOCKSWITCH_TIMEOUT_VALUE

RCC Extended TIM15 Clock Source

RCC_TIM15CLK_HCLK

RCC_TIM15CLK_PLLCLK

IS_RCC_TIM15CLKSOURCE

RCC Extended TIM16 Clock Source

RCC_TIM16CLK_HCLK

RCC_TIM16CLK_PLLCLK

IS_RCC_TIM16CLKSOURCE

RCC Extended TIM17 Clock Source

RCC_TIM17CLK_HCLK

RCC_TIM17CLK_PLLCLK

IS_RCC_TIM17CLKSOURCE

RCC Extended TIM1 Clock Source

RCC_TIM1CLK_HCLK

RCC_TIM1CLK_PLLCLK

IS_RCC_TIM1CLKSOURCE

RCC Extended TIM20 Clock Source

RCC_TIM20CLK_HCLK

RCC_TIM20CLK_PLLCLK

IS_RCC_TIM20CLKSOURCE

RCC Extended TIM2 Clock Source

RCC_TIM2CLK_HCLK

RCC_TIM2CLK_PLLCLK

IS_RCC_TIM2CLKSOURCE

RCC Extended TIM3 & TIM4 Clock Source

RCC_TIM34CLK_HCLK

RCC_TIM34CLK_PLLCLK

IS_RCC_TIM3CLKSOURCE

RCC Extended TIM8 Clock Source

RCC_TIM8CLK_HCLK

RCC_TIM8CLK_PLLCLK

IS_RCC_TIM8CLKSOURCE

RCC Extended TIMx Clock Config

__HAL_RCC_TIM1_CONFIG	**Description:** • Macro to configure the TIM1 clock (TIM1CLK).

Parameters:

- __TIM1CLKSource__: specifies the TIM1 clock source. This parameter can be one of the following values:
  - RCC_TIM1CLKSOURCE_HCLK: HCLK selected as TIM1 clock
  - RCC_TIM1CLKSOURCE_PLL: PLL Clock selected as TIM1 clock

__HAL_RCC_GET_TIM1_SOURCE

Description:

- Macro to get the TIM1 clock (TIM1CLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_TIM1CLKSOURCE_HCLK: HCLK selected as TIM1 clock
  - RCC_TIM1CLKSOURCE_PLL: PLL Clock selected as TIM1 clock

__HAL_RCC_TIM8_CONFIG

Description:

- Macro to configure the TIM8 clock (TIM8CLK).

Parameters:

- __TIM8CLKSource__: specifies the TIM8 clock source. This parameter can be one of the following values:
  - RCC_TIM8CLKSOURCE_HCLK: HCLK selected as TIM8 clock
  - RCC_TIM8CLKSOURCE_PLL: PLL Clock selected as TIM8 clock

__HAL_RCC_GET_TIM8_SOURCE

Description:

- Macro to get the TIM8 clock (TIM8CLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_TIM8CLKSOURCE_HCLK: HCLK selected as TIM8 clock
  - RCC_TIM8CLKSOURCE_PLL: PLL Clock selected as TIM8 clock

__HAL_RCC_TIM2_CONFIG

Description:

- Macro to configure the TIM2 clock (TIM2CLK).

Parameters:

- __TIM2CLKSource__: specifies the TIM2 clock source. This parameter can be one of the following values:
  - RCC_TIM2CLK_HCLK: HCLK selected as TIM2 clock
  - RCC_TIM2CLK_PLL: PLL Clock selected

as TIM2 clock

__HAL_RCC_GET_TIM2_SOURCE	**Description:** • Macro to get the TIM2 clock (TIM2CLK). **Return value:** • The: clock source can be one of the following values: – RCC_TIM2CLK_HCLK: HCLK selected as TIM2 clock – RCC_TIM2CLK_PLL: PLL Clock selected as TIM2 clock
__HAL_RCC_TIM34_CONFIG	**Description:** • Macro to configure the TIM3 & TIM4 clock (TIM34CLK). **Parameters:** • __TIM34CLKSource__: specifies the TIM3 & TIM4 clock source. This parameter can be one of the following values: – RCC_TIM34CLK_HCLK: HCLK selected as TIM3 & TIM4 clock – RCC_TIM34CLK_PLL: PLL Clock selected as TIM3 & TIM4 clock
__HAL_RCC_GET_TIM34_SOURCE	**Description:** • Macro to get the TIM3 & TIM4 clock (TIM34CLK). **Return value:** • The: clock source can be one of the following values: – RCC_TIM34CLK_HCLK: HCLK selected as TIM3 & TIM4 clock – RCC_TIM34CLK_PLL: PLL Clock selected as TIM3 & TIM4 clock
__HAL_RCC_TIM15_CONFIG	**Description:** • Macro to configure the TIM15 clock (TIM15CLK). **Parameters:** • __TIM15CLKSource__: specifies the TIM15 clock source. This parameter can be one of the following values: – RCC_TIM15CLK_HCLK: HCLK selected as TIM15 clock – RCC_TIM15CLK_PLL: PLL Clock selected as TIM15 clock
__HAL_RCC_GET_TIM15_SOURCE	**Description:** • Macro to get the TIM15 clock (TIM15CLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_TIM15CLK_HCLK: HCLK selected as TIM15 clock
  - RCC_TIM15CLK_PLL: PLL Clock selected as TIM15 clock

__HAL_RCC_TIM16_CONFIG

Description:

- Macro to configure the TIM16 clock (TIM16CLK).

Parameters:

- __TIM16CLKSource__: specifies the TIM16 clock source. This parameter can be one of the following values:
  - RCC_TIM16CLK_HCLK: HCLK selected as TIM16 clock
  - RCC_TIM16CLK_PLL: PLL Clock selected as TIM16 clock

__HAL_RCC_GET_TIM16_SOURCE

Description:

- Macro to get the TIM16 clock (TIM16CLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_TIM16CLK_HCLK: HCLK selected as TIM16 clock
  - RCC_TIM16CLK_PLL: PLL Clock selected as TIM16 clock

__HAL_RCC_TIM17_CONFIG

Description:

- Macro to configure the TIM17 clock (TIM17CLK).

Parameters:

- __TIM17CLKSource__: specifies the TIM17 clock source. This parameter can be one of the following values:
  - RCC_TIM17CLK_HCLK: HCLK selected as TIM17 clock
  - RCC_TIM17CLK_PLL: PLL Clock selected as TIM17 clock

__HAL_RCC_GET_TIM17_SOURCE

Description:

- Macro to get the TIM17 clock (TIM17CLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_TIM17CLK_HCLK: HCLK selected as TIM17 clock
  - RCC_TIM17CLK_PLL: PLL Clock

selected as TIM17 clock

__HAL_RCC_TIM20_CONFIG

Description:

- Macro to configure the TIM20 clock (TIM20CLK).

Parameters:

- __TIM20CLKSource__: specifies the TIM20 clock source. This parameter can be one of the following values:
  - RCC_TIM20CLK_HCLK: HCLK selected as TIM20 clock
  - RCC_TIM20CLK_PLL: PLL Clock selected as TIM20 clock

__HAL_RCC_GET_TIM20_SOURCE

Description:

- Macro to get the TIM20 clock (TIM20CLK).

Return value:

- The: clock source can be one of the following values:
  - RCC_TIM20CLK_HCLK: HCLK selected as TIM20 clock
  - RCC_TIM20CLK_PLL: PLL Clock selected as TIM20 clock

RCC Extended UART4 Clock Source

RCC_UART4CLKSOURCE_PCLK1

RCC_UART4CLKSOURCE_SYSCLK

RCC_UART4CLKSOURCE_LSE

RCC_UART4CLKSOURCE_HSI

IS_RCC_UART4CLKSOURCE

RCC Extended UART5 Clock Source

RCC_UART5CLKSOURCE_PCLK1

RCC_UART5CLKSOURCE_SYSCLK

RCC_UART5CLKSOURCE_LSE

RCC_UART5CLKSOURCE_HSI

IS_RCC_UART5CLKSOURCE

RCC Extended UARTx Clock Config

__HAL_RCC_UART4_CONFIG

Description:

- Macro to configure the UART4 clock (UART4CLK).

Parameters:

- __UART4CLKSource__: specifies the UART4 clock source. This parameter can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1:

PCLK1 selected as UART4 clock
- RCC_UART4CLKSOURCE_HSI: HSI selected as UART4 clock
- RCC_UART4CLKSOURCE_SYSCLK: System Clock selected as UART4 clock
- RCC_UART4CLKSOURCE_LSE: LSE selected as UART4 clock

__HAL_RCC_GET_UART4_SOURCE

Description:

- Macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1: PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI: HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_SYSCLK: System Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE: LSE selected as UART4 clock

__HAL_RCC_UART5_CONFIG

Description:

- Macro to configure the UART5 clock (UART5CLK).

Parameters:

- __UART5CLKSource__: specifies the UART5 clock source. This parameter can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1: PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI: HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_SYSCLK: System Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE: LSE selected as UART5 clock

__HAL_RCC_GET_UART5_SOURCE

Description:

- Macro to get the UART5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1: PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI: HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_SYSCLK: System Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE: LSE selected as UART5 clock

RCC Extended USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK2

RCC_USART1CLKSOURCE_SYSCLK

RCC_USART1CLKSOURCE_LSE

RCC_USART1CLKSOURCE_HSI

IS_RCC_USART1CLKSOURCE

RCC Extended USBx Clock Config

__HAL_RCC_USB_CONFIG

Description:

- Macro to configure the USB clock (USBCLK).

Parameters:

- __USBCLKSource__: specifies the USB clock source. This parameter can be one of the following values:
  - RCC_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 source.

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

RCC Extended USB Clock Source

RCC_USBPLLCLK_DIV1

RCC_USBPLLCLK_DIV1_5

IS_RCC_USBCLKSOURCE

40 HAL RTC Generic Driver

40.1 RTC Firmware driver registers structures

40.1.1 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32f3xx_hal_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutPolarity***
- ***uint32_t OutPutType***

Field Documentation

- ***uint32_t RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hour Format. This parameter can be a value of ***RTC_Hour_Formats***
- ***uint32_t RTC_InitTypeDef::AsynchPrediv***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- ***uint32_t RTC_InitTypeDef::SynchPrediv***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- ***uint32_t RTC_InitTypeDef::OutPut***
  Specifies which signal will be routed to the RTC output. This parameter can be a value of ***RTCEx_Output_selection_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***
  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutType***
  Specifies the RTC Output Pin mode. This parameter can be a value of ***RTC_Output_Type_ALARM_OUT***

40.1.2 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32f3xx_hal_rtc.h

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***
- ***uint32_t SubSeconds***
- ***uint8_t TimeFormat***

- ***uint32_t DayLightSaving***
- ***uint32_t StoreOperation***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hour. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the RTC_HourFormat_12 is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the RTC_HourFormat_24 is selected
- ***uint8_t RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint32_t RTC_TimeTypeDef::SubSeconds***
  Specifies the RTC Time SubSeconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_AM_PM_Definitions***
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***
  Specifies RTC_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of ***RTC_DayLightSaving_Definitions***
- ***uint32_t RTC_TimeTypeDef::StoreOperation***
  Specifies RTC_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of ***RTC_StoreOperation_Definitions***

40.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32f3xx_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. 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

40.1.4 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32f3xx_hal_rtc.h

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_AlarmMask_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
  Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of ***RTC_Alarm_Sub_Seconds_Masks_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of ***RTC_AlarmDateWeekDay_Definitions***
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint32_t RTC_AlarmTypeDef::Alarm***
  Specifies the alarm . This parameter can be a value of ***RTC_Alarms_Definitions***

40.1.5 RTC_HandleTypeDef

RTC_HandleTypeDef is defined in the stm32f3xx_hal_rtc.h

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***

Field Documentation

- ***RTC_TypeDef* RTC_HandleTypeDef::Instance***
  Register base address
- ***RTC_InitTypeDef RTC_HandleTypeDef::Init***
  RTC required parameters
- ***HAL_LockTypeDef RTC_HandleTypeDef::Lock***
  RTC locking object
- ***__IO HAL_RTCStateTypeDef RTC_HandleTypeDef::State***
  Time communication state

40.2 RTC Firmware driver API description

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

40.2.1 RTC Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

To allow the RTC to operate even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

1. The RTC
2. The LSE oscillator
3. PC13 to PC15 I/Os (when available)

When the backup domain is supplied by VDD (analog switch connected to VDD), the following functions are available:

1. PC14 and PC15 can be used as either GPIO or LSE pins
2. PC13 can be used as a GPIO or as the RTC_OUT pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following functions are available:

1. PC14 and PC15 can be used as LSE pins only
2. PC13 can be used as the RTC_OUT pin

40.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.

40.2.3 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

1. Enable the Power Controller (PWR) APB1 interface clock using the __PWR_CLK_ENABLE() function.
2. Enable access to RTC domain using the HAL_PWR_EnableBkUpAccess() function.
3. Select the RTC clock source using the __HAL_RCC_RTC_CONFIG() function.
4. Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() function.

40.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format 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.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTC_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTC_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTC_GetWakeUpTimer() function.

TimeStamp configuration

- Configure the RTC_AF trigger and enables the RTC TimeStamp using the HAL_RTC_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTC_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTC_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and Configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, precharge or discharge and Pull-UP using the HAL_RTC_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTC_SetTamper_IT() function.

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTC_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTC_BKUPRead() function.

40.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.

40.2.6 Initialization and de-initialization functions

This section provide functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
   - A 7-bit asynchronous prescaler and A 15-bit synchronous prescaler.
   - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To Configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

- ***HAL_RTC_Init()***
- ***HAL_RTC_DeInit()***
- ***HAL_RTC_MspInit()***
- ***HAL_RTC_MspDeInit()***

40.2.7 RTC Time and Date functions

This section provide functions allowing to configure Time and Date features

- *HAL_RTC_SetTime()*
- *HAL_RTC_GetTime()*
- *HAL_RTC_SetDate()*
- *HAL_RTC_GetDate()*

40.2.8 RTC Alarm functions

This section provide functions allowing to configure Alarm feature

- *HAL_RTC_SetAlarm()*
- *HAL_RTC_SetAlarm_IT()*
- *HAL_RTC_DeactivateAlarm()*
- *HAL_RTC_GetAlarm()*
- *HAL_RTC_AlarmIRQHandler()*
- *HAL_RTC_AlarmAEventCallback()*
- *HAL_RTC_PollForAlarmAEvent()*

40.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization
- *HAL_RTC_WaitForSynchro()*

40.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state
- *HAL_RTC_GetState()*

40.2.11 HAL_RTC_Init

Function Name	**HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)**
Function Description	Initializes the RTC peripheral.
Parameters	• **hrtc:** RTC handle
Return values	• HAL status

40.2.12 HAL_RTC_DeInit

Function Name	**HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)**

Function Description	DeInitializes the RTC peripheral.
Parameters	• **hrtc:** RTC handle
Return values	• HAL status
Notes	• This function doesn't reset the RTC Backup Data registers.

40.2.13 HAL_RTC_MspInit

Function Name	**void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)**
Function Description	Initializes the RTC MSP.
Parameters	• **hrtc:** RTC handle
Return values	• None

40.2.14 HAL_RTC_MspDeInit

Function Name	**void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)**
Function Description	DeInitializes the RTC MSP.
Parameters	• **hrtc:** RTC handle
Return values	• None

40.2.15 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:** RTC handle • **sTime:** Pointer to Time structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format
Return values	• HAL status

40.2.16 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:** RTC handle • **sTime:** Pointer to Time structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN:

		Binary data format FORMAT_BCD: BCD data format
Return values	•	HAL status
Notes	•	Call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers.

40.2.17 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:** RTC handle • **sDate:** Pointer to date structure • **Format:** specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format
Return values	• HAL status

40.2.18 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:** RTC handle • **sDate:** Pointer to Date structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN : Binary data format FORMAT_BCD : BCD data format
Return values	• HAL status

40.2.19 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:** RTC handle • **sAlarm:** Pointer to Alarm structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format
Return values	• HAL status

40.2.20 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:** RTC handle • **sAlarm:** Pointer to Alarm structure • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format
Return values	• HAL status
Notes	• The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()). • The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

40.2.21 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:** RTC handle • **Alarm:** Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A : AlarmA RTC_ALARM_B : AlarmB
Return values	• HAL status

40.2.22 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:** RTC handle • **sAlarm:** Pointer to Date structure • **Alarm:** Specifies the Alarm This parameter can be one of the following values: RTC_ALARM_A: AlarmA RTC_ALARM_B: AlarmB • **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format
Return values	• HAL status

40.2.23 HAL_RTC_AlarmIRQHandler

Function Name	**void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)**
Function Description	This function handles Alarm interrupt request.
Parameters	• **hrtc:** RTC handle
Return values	• None

40.2.24 HAL_RTC_AlarmAEventCallback

Function Name	**void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Alarm A callback.
Parameters	• **hrtc:** RTC handle
Return values	• None

40.2.25 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:** RTC handle • **Timeout:** Timeout duration
Return values	• HAL status

40.2.26 HAL_RTC_WaitForSynchro

Function Name	**HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)**
Function Description	Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	• **hrtc:** RTC handle
Return values	• HAL status
Notes	• The RTC Resynchronization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function. • To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow

registers.

40.2.27 HAL_RTC_GetState

Function Name	**HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)**
Function Description	Returns the Alarm state.
Parameters	• **hrtc:** RTC handle
Return values	• HAL state

40.3 RTC Firmware driver defines

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

40.3.1 RTC

RTC

RTC AlarmDateWeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

RTC AlarmMask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

IS_ALARM_MASK

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

IS_ALARM

RTC Alarm Definitions

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
RTC_ALARMSUBSECONDMASK_SS14_1	SS[14:1] are ignored in Alarm comparison.

	Only SS[0] is compared.
RTC_ALARMSUBSECONDMASK_SS14_2	SS[14:2] are ignored in Alarm comparison. Only SS[1:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_3	SS[14:3] are ignored in Alarm comparison. Only SS[2:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_4	SS[14:4] are ignored in Alarm comparison. Only SS[3:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_5	SS[14:5] are ignored in Alarm comparison. Only SS[4:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_6	SS[14:6] are ignored in Alarm comparison. Only SS[5:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_7	SS[14:7] are ignored in Alarm comparison. Only SS[6:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_8	SS[14:8] are ignored in Alarm comparison. Only SS[7:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_9	SS[14:9] are ignored in Alarm comparison. Only SS[8:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_10	SS[14:10] are ignored in Alarm comparison. Only SS[9:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_11	SS[14:11] are ignored in Alarm comparison. Only SS[10:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_12	SS[14:12] are ignored in Alarm comparison.Only SS[11:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_13	SS[14:13] are ignored in Alarm comparison. Only SS[12:0] are compared
RTC_ALARMSUBSECONDMASK_SS14	SS[14] is don't care in Alarm comparison.Only SS[13:0] are compared
RTC_ALARMSUBSECONDMASK_None	SS[14:0] are compared and must match to activate alarm.
IS_RTC_ALARM_SUB_SECOND_MASK	

RTC Alarm Sub Seconds Value

IS_RTC_ALARM_SUB_SECOND_VALUE

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

IS_RTC_HOURFORMAT12

RTC Asynchronous Predivider

IS_RTC_ASYNCH_PREDIV

RTC DayLightSaving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

IS_RTC_DAYLIGHT_SAVING

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

Parameters:

- __HANDLE__: RTC handle.

Return value:

- None:

__HAL_RTC_WRITEPROTECTION_DISABLE

Description:

- Disable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_WRITEPROTECTION_ENABLE

Description:

- Enable the write protection for RTC registers.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_ALARMA_ENABLE

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_ALARMA_DISABLE

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_ALARMB_ENABLE

Description:

- Enable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_ALARMB_DISABLE

Description:

- Disable the RTC ALARMB peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_ALARM_ENABLE_IT

Description:

- Enable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA: Alarm A interrupt
  - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None:

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA: Alarm A interrupt
  - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None:

__HAL_RTC_ALARM_GET_IT

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm

interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_ALRA: Alarm A interrupt
  - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None:

__HAL_RTC_ALARM_GET_FLAG

Description:

- Get the selected RTC Alarm's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF
  - RTC_FLAG_ALRAWF
  - RTC_FLAG_ALRBWF

Return value:

- None:

__HAL_RTC_ALARM_CLEAR_FLAG

Description:

- Clear the RTC Alarm's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- None:

RTC_EXTI_LINE_ALARM_EVENT — External interrupt line 17 Connected to the RTC Alarm event

RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT — External interrupt line 19 Connected to the RTC Tamper and Time Stamp events

RTC_EXTI_LINE_WAKEUPTIMER_EVENT — External interrupt line 20 Connected to the RTC Wakeup event

__HAL_RTC_EXTI_ENABLE_IT

Description:

- Enable the RTC Exti line.

Parameters:

- __EXTILINE__: specifies the RTC Exti sources to be enabled or disabled. This parameter can be:
  - RTC_EXTI_LINE_ALARM_EVENT

- RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT
- RTC_EXTI_LINE_WAKEUPTIMER_EVENT

Return value:

- None:

__HAL_RTC_ENABLE_IT

__HAL_RTC_EXTI_DISABLE_IT

Description:

- Disable the RTC Exti line.

Parameters:

- __EXTILINE__: specifies the RTC Exti sources to be enabled or disabled. This parameter can be:
  - RTC_EXTI_LINE_ALARM_EVENT
  - RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT
  - RTC_EXTI_LINE_WAKEUPTIMER_EVENT

Return value:

- None:

__HAL_RTC_DISABLE_IT

__HAL_RTC_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTILINE__: specifies the RTC Exti sources to be enabled or disabled. This parameter can be:
  - RTC_EXTI_LINE_ALARM_EVENT
  - RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT
  - RTC_EXTI_LINE_WAKEUPTIMER_EVENT

Return value:

- None:

__HAL_RTC_EXTI_CLEAR_FLAG

Description:

- Clear the RTC Exti flags.

Parameters:

- __FLAG__: specifies the RTC Exti sources to be enabled or disabled. This parameter can be:
  - RTC_EXTI_LINE_ALARM_EVENT
  - RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT

– RTC_EXTI_LINE_WAKEUPTIMER_EVENT

Return value:

- None:

__HAL_RTC_CLEAR_FLAG

RTC Flags Definitions

RTC_FLAG_RECALPF

RTC_FLAG_TAMP3F

RTC_FLAG_TAMP2F

RTC_FLAG_TAMP1F

RTC_FLAG_TSOVF

RTC_FLAG_TSF

RTC_FLAG_WUTF

RTC_FLAG_ALRBF

RTC_FLAG_ALRAF

RTC_FLAG_INITF

RTC_FLAG_RSF

RTC_FLAG_INITS

RTC_FLAG_SHPF

RTC_FLAG_WUTWF

RTC_FLAG_ALRBWF

RTC_FLAG_ALRAWF

RTC Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

IS_RTC_HOUR_FORMAT

RTC Input parameter format definitions

FORMAT_BIN

FORMAT_BCD

IS_RTC_FORMAT

RTC Interrupts Definitions

RTC_IT_TS

RTC_IT_WUT

RTC_IT_ALRB

RTC_IT_ALRA

RTC_IT_TAMP

RTC_IT_TAMP1

RTC_IT_TAMP2

RTC_IT_TAMP3

RTC Mask Definition

RTC_TR_RESERVED_MASK

RTC_DR_RESERVED_MASK

RTC_INIT_MASK

RTC_RSF_MASK

RTC_FLAGS_MASK

RTC_TIMEOUT_VALUE

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

IS_RTC_MONTH

IS_RTC_DATE

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

IS_RTC_OUTPUT_POL

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_OPENDRAIN

RTC_OUTPUT_TYPE_PUSHPULL

IS_RTC_OUTPUT_TYPE

RTC StoreOperation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

IS_RTC_STORE_OPERATION

RTC Synchronous Predivider

IS_RTC_SYNCH_PREDIV

RTC Time Definitions

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

IS_RTC_WEEKDAY

RTC Year Date Definitions

IS_RTC_YEAR

41 HAL RTC Extension Driver

41.1 RTCEx Firmware driver registers structures

41.1.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the stm32f3xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Trigger***
- ***uint32_t Filter***
- ***uint32_t SamplingFrequency***
- ***uint32_t PrechargeDuration***
- ***uint32_t TamperPullUp***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of ***RTCEx_Tamper_Pins_Definitions***
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of ***RTCEx_Tamper_Trigger_Definitions***
- ***uint32_t RTC_TamperTypeDef::Filter***
  Specifies the RTC Filter Tamper. This parameter can be a value of ***RTCEx_Tamper_Filter_Definitions***
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
  Specifies the sampling frequency. This parameter can be a value of ***RTCEx_Tamper_Sampling_Frequencies_Definitions***
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
  Specifies the Precharge Duration . This parameter can be a value of ***RTCEx_Tamper_Pin_Precharge_Duration_Definitions***
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
  Specifies the Tamper PullUp . This parameter can be a value of ***RTCEx_Tamper_Pull_UP_Definitions***
- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection_Definitions***

41.2 RTCEx Firmware driver API description

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

41.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEx_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEx_GetWakeUpTimer() function.

TimeStamp configuration

- Configure the RTC_AFx trigger and enables the RTC TimeStamp using the HAL_RTCEx_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEx_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.
- The TIMESTAMP alternate function is mapped to RTC_AF1 (PC13).

Tamper configuration

- Enable the RTC Tamper and Configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, precharge or discharge and Pull-UP using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The TAMPER1 alternate function is mapped to RTC_AF1 (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.

41.2.2 RTC TimeStamp and Tamper functions

This section provide functions allowing to configure TimeStamp feature

- ***HAL_RTCEx_SetTimeStamp()***
- ***HAL_RTCEx_SetTimeStamp_IT()***
- ***HAL_RTCEx_DeactivateTimeStamp()***
- ***HAL_RTCEx_GetTimeStamp()***
- ***HAL_RTCEx_SetTamper()***
- ***HAL_RTCEx_SetTamper_IT()***
- ***HAL_RTCEx_DeactivateTamper()***
- ***HAL_RTCEx_TamperTimeStampIRQHandler()***

- *HAL_RTCEx_TimeStampEventCallback()*
- *HAL_RTCEx_Tamper1EventCallback()*
- *HAL_RTCEx_Tamper2EventCallback()*
- *HAL_RTCEx_Tamper3EventCallback()*
- *HAL_RTCEx_PollForTimeStampEvent()*
- *HAL_RTCEx_PollForTamper1Event()*
- *HAL_RTCEx_PollForTamper2Event()*
- *HAL_RTCEx_PollForTamper3Event()*

41.2.3 RTC Wake-up functions

This section provide functions allowing to configure Wake-up feature

- *HAL_RTCEx_SetWakeUpTimer()*
- *HAL_RTCEx_SetWakeUpTimer_IT()*
- *HAL_RTCEx_DeactivateWakeUpTimer()*
- *HAL_RTCEx_GetWakeUpTimer()*
- *HAL_RTCEx_WakeUpTimerIRQHandler()*
- *HAL_RTCEx_WakeUpTimerEventCallback()*
- *HAL_RTCEx_PollForWakeUpTimerEvent()*

41.2.4 Extended 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 Coarse calibration parameters.
- Deactivates the Coarse calibration parameters
- Sets the Smooth calibration parameters.
- Configures the Synchronization Shift Control Settings.
- Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enables the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enables the Bypass Shadow feature.
- Disables the Bypass Shadow feature.
- *HAL_RTCEx_BKUPWrite()*
- *HAL_RTCEx_BKUPRead()*
- *HAL_RTCEx_SetSmoothCalib()*
- *HAL_RTCEx_SetSynchroShift()*
- *HAL_RTCEx_SetCalibrationOutPut()*
- *HAL_RTCEx_DeactivateCalibrationOutPut()*
- *HAL_RTCEx_SetRefClock()*
- *HAL_RTCEx_DeactivateRefClock()*
- *HAL_RTCEx_EnableBypassShadow()*
- *HAL_RTCEx_DisableBypassShadow()*

41.2.5 Extended features functions

This section provides functions allowing to:

- RTC Alram B callback
- RTC Poll for Alarm B request
- *HAL_RTCEx_AlarmBEventCallback()*
- *HAL_RTCEx_PollForAlarmBEvent()*

41.2.6 HAL_RTCEx_SetTimeStamp

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)**
Function Description	Sets TimeStamp.
Parameters	• **hrtc:** RTC handle • **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following: TimeStampEdge_Rising: the Time stamp event occurs on the rising edge of the related pin. TimeStampEdge_Falling: the Time stamp event occurs on the falling edge of the related pin. • **RTC_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_PC13: PC13 is selected as RTC TimeStamp Pin.
Return values	• HAL status
Notes	• This API must be called before enabling the TimeStamp feature.

41.2.7 HAL_RTCEx_SetTimeStamp_IT

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)**
Function Description	Sets TimeStamp with Interrupt.
Parameters	• **hrtc:** RTC handle • **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following: TimeStampEdge_Rising: the Time stamp event occurs on the rising edge of the related pin. TimeStampEdge_Falling: the Time stamp event occurs on the falling edge of the related pin. • **RTC_TimeStampPin:** Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_PC13: PC13 is selected as RTC TimeStamp Pin.
Return values	• HAL status
Notes	• This API must be called before enabling the TimeStamp feature.

41.2.8 HAL_RTCEx_DeactivateTimeStamp

Function Name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)**
Function Description	Deactivates TimeStamp.
Parameters	• **hrtc:** RTC handle
Return values	• HAL status

41.2.9 HAL_RTCEx_GetTimeStamp

Function Name	**HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)**
Function Description	Gets the RTC TimeStamp value.
Parameters	• **hrtc:** RTC handle • **sTimeStamp:** Pointer to Time structure • **sTimeStampDate:** Pointer to Date structure • **Format:** specifies the format of the entered parameters. This parameter can be one of the following values: Format_BIN: Binary data format Format_BCD: BCD data format
Return values	• HAL status

41.2.10 HAL_RTCEx_SetTamper

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)**
Function Description	Sets Tamper.
Parameters	• **hrtc:** RTC handle • **sTamper:** Pointer to Tamper Structure.
Return values	• HAL status
Notes	• By calling this API we disable the tamper interrupt for all tampers.

41.2.11 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:** RTC handle • **sTamper:** Pointer to RTC Tamper.

Return values	• HAL status
Notes	• By calling this API we force the tamper interrupt for all tampers.

41.2.12 HAL_RTCEx_DeactivateTamper

Function Name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)**
Function Description	Deactivates Tamper.
Parameters	• **hrtc:** RTC handle • **Tamper:** Selected tamper pin. This parameter can be any combination of RTC_TAMPER_1, RTC_TAMPER_2 and RTC_TAMPER_3.
Return values	• HAL status

41.2.13 HAL_RTCEx_TamperTimeStampIRQHandler

Function Name	**void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)**
Function Description	This function handles TimeStamp interrupt request.
Parameters	• **hrtc:** RTC handle
Return values	• None

41.2.14 HAL_RTCEx_TimeStampEventCallback

Function Name	**void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	TimeStamp callback.
Parameters	• **hrtc:** RTC handle
Return values	• None

41.2.15 HAL_RTCEx_Tamper1EventCallback

Function Name	**void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Tamper 1 callback.
Parameters	• **hrtc:** RTC handle
Return values	• None

41.2.16 HAL_RTCEx_Tamper2EventCallback

Function Name	**void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Tamper 2 callback.
Parameters	• **hrtc:** RTC handle
Return values	• None

41.2.17 HAL_RTCEx_Tamper3EventCallback

Function Name	**void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Tamper 3 callback.
Parameters	• **hrtc:** RTC handle
Return values	• None

41.2.18 HAL_RTCEx_PollForTimeStampEvent

Function Name	**HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function Description	This function handles TimeStamp polling request.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• HAL status

41.2.19 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:** RTC handle • **Timeout:** Timeout duration
Return values	• HAL status

41.2.20 HAL_RTCEx_PollForTamper2Event

Function Name	**HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function Description	This function handles Tamper2 Polling.

Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• HAL status

41.2.21 HAL_RTCEx_PollForTamper3Event

Function Name	**HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function Description	This function handles Tamper3 Polling.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• HAL status

41.2.22 HAL_RTCEx_SetWakeUpTimer

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)**
Function Description	Sets wake up timer.
Parameters	• **hrtc:** RTC handle • **WakeUpCounter:** Wake up counter • **WakeUpClock:** Wake up clock
Return values	• HAL status

41.2.23 HAL_RTCEx_SetWakeUpTimer_IT

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)**
Function Description	Sets wake up timer with interrupt.
Parameters	• **hrtc:** RTC handle • **WakeUpCounter:** wake up counter • **WakeUpClock:** wake up clock
Return values	• HAL status

41.2.24 HAL_RTCEx_DeactivateWakeUpTimer

Function Name	**uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)**
Function Description	Deactivates wake up timer counter.
Parameters	• **hrtc:** RTC handle

Return values	• HAL status

41.2.25 HAL_RTCEx_GetWakeUpTimer

Function Name	**uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)**
Function Description	Gets wake up timer counter.
Parameters	• **hrtc:** RTC handle
Return values	• Counter value

41.2.26 HAL_RTCEx_WakeUpTimerIRQHandler

Function Name	**void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)**
Function Description	This function handles Wake Up Timer interrupt request.
Parameters	• **hrtc:** RTC handle
Return values	• None

41.2.27 HAL_RTCEx_WakeUpTimerEventCallback

Function Name	**void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Wake Up Timer callback.
Parameters	• **hrtc:** RTC handle
Return values	• None

41.2.28 HAL_RTCEx_PollForWakeUpTimerEvent

Function Name	**HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function Description	This function handles Wake Up Timer Polling.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• HAL status

41.2.29 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:** RTC handle • **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register. • **Data:** Data to be written in the specified RTC Backup data register.
Return values	• None

41.2.30 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:** RTC handle • **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.
Return values	• Read value

41.2.31 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:** Select the Smooth Calibration Period. This parameter can be can be one of the following values : RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration periode is 32s. RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration periode is 16s. RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibartion periode is 8s. • **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values: RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK puls every 2*11 pulses. RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added. • **SmouthCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.
Return values	• HAL status

Notes	•	To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmouthCalibMinusPulsesValue mut be equal to 0.

41.2.32 HAL_RTCEx_SetSynchroShift

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)**
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	• **hrtc:** RTC handle • **ShiftAdd1S:** Select to add or not 1 second to the time calendar. This parameter can be one of the following values : RTC_SHIFTADD1S_SET: Add one second to the clock calendar. RTC_SHIFTADD1S_RESET: No effect. • **ShiftSubFS:** Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.
Return values	• HAL status
Notes	• When REFCKON is set, firmware must not write to Shift control register.

41.2.33 HAL_RTCEx_SetCalibrationOutPut

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)**
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	• **hrtc:** RTC handle • **CalibOutput:** Select the Calibration output Selection . This parameter can be one of the following values: RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz. RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.
Return values	• HAL status

41.2.34 HAL_RTCEx_DeactivateCalibrationOutPut

Function Name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)**
Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	• **hrtc:** RTC handle

Return values	• HAL status

41.2.35 HAL_RTCEx_SetRefClock

Function Name	**HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)**
Function Description	Enables the RTC reference clock detection.
Parameters	• **hrtc:** RTC handle
Return values	• HAL status

41.2.36 HAL_RTCEx_DeactivateRefClock

Function Name	**HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)**
Function Description	Disable the RTC reference clock detection.
Parameters	• **hrtc:** RTC handle
Return values	• HAL status

41.2.37 HAL_RTCEx_EnableBypassShadow

Function Name	**HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)**
Function Description	Enables the Bypass Shadow feature.
Parameters	• **hrtc:** RTC handle
Return values	• HAL status
Notes	• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

41.2.38 HAL_RTCEx_DisableBypassShadow

Function Name	**HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)**
Function Description	Disables the Bypass Shadow feature.
Parameters	• **hrtc:** RTC handle
Return values	• HAL status
Notes	• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

41.2.39 HAL_RTCEx_AlarmBEventCallback

Function Name	**void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)**
Function Description	Alarm B callback.
Parameters	• **hrtc:** RTC handle
Return values	• None

41.2.40 HAL_RTCEx_PollForAlarmBEvent

Function Name	**HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)**
Function Description	This function handles AlarmB Polling request.
Parameters	• **hrtc:** RTC handle • **Timeout:** Timeout duration
Return values	• HAL status

41.3 RTCEx Firmware driver defines

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

41.3.1 RTCEx

RTCEx

RTC Extended Add 1 Second Parameter Definition

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

IS_RTC_SHIFT_ADD1S

RTC Extended Backup Registers Definition

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC_BKP_DR5

RTC_BKP_DR6

RTC_BKP_DR7

RTC_BKP_DR8

RTC_BKP_DR9

RTC_BKP_DR10

RTC_BKP_DR11

RTC_BKP_DR12

RTC_BKP_DR13

RTC_BKP_DR14

RTC_BKP_DR15

IS_RTC_BKP

RTC Extended Calib Output selection Definition

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

IS_RTC_CALIB_OUTPUT

RTC Extended Exported Macros

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_TIMESTAMP_ENABLE

Description:

- Enable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_TIMESTAMP_DISABLE

Description:

- Disable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_CALIBRATION_OUTPUT_ENABLE

Description:

- Enable the RTC calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_CALIBRATION_OUTPUT_DISABLE

Description:

- Disable the calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_CLOCKREF_DETECTION_ENABLE

Description:

- Enable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_CLOCKREF_DETECTION_DISABLE

Description:

- Disable the clock reference detection.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None:

__HAL_RTC_TIMESTAMP_ENABLE_IT

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.

- __INTERRUPT__: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_TS: TimeStamp interrupt

Return value:

- None:

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_WUT: WakeUpTimer A interrupt

Return value:

- None:

__HAL_RTC_TIMESTAMP_DISABLE_IT

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_TS: TimeStamp interrupt

Return value:

- None:

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.

- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_WUT: WakeUpTimer A interrupt

Return value:

- None:

__HAL_RTC_TAMPER_GET_IT

Description:

- Check whether the specified RTC Tamper interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_TAMP1

Return value:

- None:

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_WUT: WakeUpTimer A interrupt

Return value:

- None:

__HAL_RTC_TIMESTAMP_GET_IT

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.

- __FLAG__: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_TS: TimeStamp interrupt

Return value:

- None:

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:

- Get the selected RTC TimeStamp's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_TSF
  - RTC_FLAG_TSOVF

Return value:

- None:

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:

- Get the selected RTC WakeUpTimer's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_WUTF
  - RTC_FLAG_WUTWF

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_SHIFT_GET_FLAG

Description:

- Get the selected RTC shift operation's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  - RTC_FLAG_SHPF

Return value:

- None:

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamp's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - RTC_FLAG_TSF

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_WAKEUPTIMER_CLEAR_FLAG

Description:

- Clear the RTC Wake Up timer'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_WUTF

Return value:

- None:

RTC Extended Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARMB

RTC_OUTPUT_WAKEUP

IS_RTC_OUTPUT

RTC Extended Smooth calib Minus pulses Definition

IS_RTC_SMOOTH_CALIB_MINUS

RTC Extended Smooth calib period Definition

RTC_SMOOTHCALIB_PERIOD_32SEC	If RTCCLK = 32768 Hz, Smooth calibation period is 32s, else 2exp20 RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_16SEC	If RTCCLK = 32768 Hz, Smooth calibation period is 16s, else 2exp19 RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_8SEC	If RTCCLK = 32768 Hz, Smooth calibation period is 8s, else 2exp18 RTCCLK seconds

IS_RTC_SMOOTH_CALIB_PERIOD

RTC Extended Smooth calib Plus pulses Definition

RTC_SMOOTHCALIB_PLUSPULSES_SET	The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8
RTC_SMOOTHCALIB_PLUSPULSES_RESET	The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0]

IS_RTC_SMOOTH_CALIB_PLUS

RTC Extended Substract Fraction Of Second Value

IS_RTC_SHIFT_SUBFS

RTC Extended Tamper Filter Definition

RTC_TAMPERFILTER_DISABLE	Tamper filter is disabled

RTC_TAMPERFILTER_2SAMPLE	Tamper is activated after 2 consecutive samples at the active level
RTC_TAMPERFILTER_4SAMPLE	Tamper is activated after 4 consecutive samples at the active level
RTC_TAMPERFILTER_8SAMPLE	Tamper is activated after 8 consecutive samples at the active level.

IS_TAMPER_FILTER

RTC Extended Tamper Pins Definition

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

IS_TAMPER

RTC Extended Tamper Pin Precharge Duration Definition

RTC_TAMPERPRECHARGEDURATION_1RTCCLK	Tamper pins are pre-charged before sampling during 1 RTCCLK cycle
RTC_TAMPERPRECHARGEDURATION_2RTCCLK	Tamper pins are pre-charged before sampling during 2 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_4RTCCLK	Tamper pins are pre-charged before sampling during 4 RTCCLK cycles
RTC_TAMPERPRECHARGEDURATION_8RTCCLK	Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

IS_TAMPER_PRECHARGE_DURATION

RTC Extended Tamper Pull UP Definition

RTC_TAMPER_PULLUP_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TAMPER_PULLUP_DISABLE	TimeStamp on Tamper Detection event is not saved

IS_TAMPER_PULLUP_STATE

RTC Extended Tamper Sampling Frequencies Definition

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192	Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

IS_TAMPER_SAMPLING_FREQ

RTC Extended Tamper TimeStampOnTamperDetection Definition

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TIMESTAMPONTAMPERDETECTION_DISABLE	TimeStamp on Tamper Detection event is not saved

IS_TAMPER_TIMESTAMPONTAMPER_DETECTION

RTC Extended Tamper Trigger Definition

RTC_TAMPERTRIGGER_RISINGEDGE

RTC_TAMPERTRIGGER_FALLINGEDGE

RTC_TAMPERTRIGGER_LOWLEVEL

RTC_TAMPERTRIGGER_HIGHLEVEL

IS_TAMPER_TRIGGER

RTC Extended TimeStamp Pin Selection

RTC_TIMESTAMPPIN_PC13

IS_RTC_TIMESTAMP_PIN

RTC Extended Time Stamp Edges definition

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

IS_TIMESTAMP_EDGE

RTC Extended Wakeup Timer Definition

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

IS_WAKEUP_CLOCK

IS_WAKEUP_COUNTER

42 HAL SDADC Generic Driver

42.1 SDADC Firmware driver registers structures

42.1.1 SDADC_InitTypeDef

SDADC_InitTypeDef is defined in the stm32f3xx_hal_sdadc.h

Data Fields

- ***uint32_t IdleLowPowerMode***
- ***uint32_t FastConversionMode***
- ***uint32_t SlowClockMode***
- ***uint32_t ReferenceVoltage***

Field Documentation

- ***uint32_t SDADC_InitTypeDef::IdleLowPowerMode***
  Specifies if SDADC can enter in power down or standby when idle. This parameter can be a value of ***SDADC_Idle_Low_Power_Mode***
- ***uint32_t SDADC_InitTypeDef::FastConversionMode***
  Specifies if Fast conversion mode is enabled or not. This parameter can be a value of ***SDADC_Fast_Conv_Mode***
- ***uint32_t SDADC_InitTypeDef::SlowClockMode***
  Specifies if slow clock mode is enabled or not. This parameter can be a value of ***SDADC_Slow_Clock_Mode***
- ***uint32_t SDADC_InitTypeDef::ReferenceVoltage***
  Specifies the reference voltage. This parameter can be a value of ***SDADC_Reference_Voltage***

42.1.2 SDADC_HandleTypeDef

SDADC_HandleTypeDef is defined in the stm32f3xx_hal_sdadc.h

Data Fields

- ***SDADC_TypeDef * Instance***
- ***SDADC_InitTypeDef Init***
- ***DMA_HandleTypeDef * hdma***
- ***uint32_t RegularContMode***
- ***uint32_t InjectedContMode***
- ***uint32_t InjectedChannelsNbr***
- ***uint32_t InjConvRemaining***
- ***uint32_t RegularTrigger***
- ***uint32_t InjectedTrigger***
- ***uint32_t ExtTriggerEdge***
- ***uint32_t RegularMultimode***
- ***uint32_t InjectedMultimode***
- ***HAL_SDADC_StateTypeDef State***

- ***uint32_t ErrorCode***

Field Documentation

- ***SDADC_TypeDef* SDADC_HandleTypeDef::Instance***
  SDADC registers base address
- ***SDADC_InitTypeDef SDADC_HandleTypeDef::Init***
  SDADC init parameters
- ***DMA_HandleTypeDef* SDADC_HandleTypeDef::hdma***
  SDADC DMA Handle parameters
- ***uint32_t SDADC_HandleTypeDef::RegularContMode***
  Regular conversion continuous mode
- ***uint32_t SDADC_HandleTypeDef::InjectedContMode***
  Injected conversion continuous mode
- ***uint32_t SDADC_HandleTypeDef::InjectedChannelsNbr***
  Number of channels in injected sequence
- ***uint32_t SDADC_HandleTypeDef::InjConvRemaining***
  Injected conversion remaining
- ***uint32_t SDADC_HandleTypeDef::RegularTrigger***
  Current trigger used for regular conversion
- ***uint32_t SDADC_HandleTypeDef::InjectedTrigger***
  Current trigger used for injected conversion
- ***uint32_t SDADC_HandleTypeDef::ExtTriggerEdge***
  Rising, falling or both edges selected
- ***uint32_t SDADC_HandleTypeDef::RegularMultimode***
  current type of regular multimode
- ***uint32_t SDADC_HandleTypeDef::InjectedMultimode***
  Current type of injected multimode
- ***HAL_SDADC_StateTypeDef SDADC_HandleTypeDef::State***
  SDADC state
- ***uint32_t SDADC_HandleTypeDef::ErrorCode***
  SDADC Error code

42.1.3 SDADC_ConfParamTypeDef

SDADC_ConfParamTypeDef is defined in the stm32f3xx_hal_sdadc.h

Data Fields

- ***uint32_t InputMode***
- ***uint32_t Gain***
- ***uint32_t CommonMode***
- ***uint32_t Offset***

Field Documentation

- ***uint32_t SDADC_ConfParamTypeDef::InputMode***
  Specifies the input mode (single ended, differential...) This parameter can be any value of ***SDADC_InputMode***
- ***uint32_t SDADC_ConfParamTypeDef::Gain***
  Specifies the gain setting. This parameter can be any value of ***SDADC_Gain***

- ***uint32_t SDADC_ConfParamTypeDef::CommonMode***
  Specifies the common mode setting (VSSA, VDDA, VDDA/2). This parameter can be any value of ***SDADC_CommonMode***
- ***uint32_t SDADC_ConfParamTypeDef::Offset***
  Specifies the 12-bit offset value. This parameter can be any value lower or equal to 0x00000FFF

42.2 SDADC Firmware driver API description

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

42.2.1 SDADC specific features

1. 16-bit sigma delta architecture.
2. Self calibration.
3. Interrupt generation at the end of calibration, regular/injected conversion and in case of overrun events.
4. Single and continuous conversion modes.
5. External trigger option with configurable polarity for injected conversion.
6. Multi mode (synchronized another SDADC with SDADC1).
7. DMA request generation during regular or injected channel conversion.

42.2.2 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_SDADC_MspInit() :
   - Enable SDADCx clock interface with __SDADCx_CLK_ENABLE().
   - Configure SDADCx clock divider with HAL_RCCEx_PeriphCLKConfig.
   - Enable power on SDADC with HAL_PWREx_EnableSDADCAnalog().
   - Enable the clocks for the SDADC GPIOS with __GPIOx_CLK_ENABLE().
   - Configure these SDADC pins in analog mode using HAL_GPIO_Init().
   - If interrupt mode is used, enable and configure SDADC global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
   - If DMA mode is used, configure DMA with HAL_DMA_Init and link it with SDADC handle using __HAL_LINKDMA.
2. Configure the SDADC low power mode, fast conversion mode, slow clock mode and SDADC1 reference voltage using the HAL_ADC_Init() function. If multiple SDADC are used, please configure first SDADC1 with the common reference voltage.
3. Prepare channel configurations (input mode, common mode, gain and offset) using HAL_SDADC_PrepareChannelConfig and associate channel with one configuration using HAL_SDADC_AssociateChannelConfig.

Calibration

1. Start calibration using HAL_SDADC_StartCalibration or HAL_SDADC_CalibrationStart_IT.
2. In polling mode, use HAL_SDADC_PollForCalibEvent to detect the end of calibration.
3. In interrupt mode, HAL_SDADC_CalibrationCpltCallback will be called at the end of calibration.

Regular channel conversion

1. Select trigger for regular conversion using HAL_SDADC_SelectRegularTrigger.
2. Select regular channel and enable/disable continuous mode using HAL_SDADC_ConfigChannel.
3. Start regular conversion using HAL_SDADC_Start, HAL_SDADC_Start_IT or HAL_SDADC_Start_DMA.
4. In polling mode, use HAL_SDADC_PollForConversion to detect the end of regular conversion.
5. In interrupt mode, HAL_SDADC_ConvCpltCallback will be called at the end of regular conversion.
6. Get value of regular conversion using HAL_SDADC_GetValue.
7. In DMA mode, HAL_SDADC_ConvHalfCpltCallback and HAL_SDADC_ConvCpltCallback will be called respectively at the half transfer and at the transfer complete.
8. Stop regular conversion using HAL_SDADC_Stop, HAL_SDADC_Stop_IT or HAL_SDADC_Stop_DMA.

Injected channels conversion

1. Enable/disable delay on injected conversion using HAL_SDADC_SelectInjectedDelay.
2. If external trigger is used for injected conversion, configure this trigger using HAL_SDADC_SelectInjectedExtTrigger.
3. Select trigger for injected conversion using HAL_SDADC_SelectInjectedTrigger.
4. Select injected channels and enable/disable continuous mode using HAL_SDADC_InjectedConfigChannel.
5. Start injected conversion using HAL_SDADC_InjectedStart, HAL_SDADC_InjectedStart_IT or HAL_SDADC_InjectedStart_DMA.
6. In polling mode, use HAL_SDADC_PollForInjectedConversion to detect the end of injected conversion.
7. In interrupt mode, HAL_SDADC_InjectedConvCpltCallback will be called at the end of injected conversion.
8. Get value of injected conversion and corresponding channel using HAL_SDADC_InjectedGetValue.
9. In DMA mode, HAL_SDADC_InjectedConvHalfCpltCallback and HAL_SDADC_InjectedConvCpltCallback will be called respectively at the half transfer and at the transfer complete.
10. Stop injected conversion using HAL_SDADC_InjectedStop, HAL_SDADC_InjectedStop_IT or HAL_SDADC_InjectedStop_DMA.

Multi mode regular channels conversions

1. Select type of multimode (SDADC1/SDADC2 or SDADC1/SDADC3) using HAL_SDADC_MultiModeConfigChannel.

2. Select software trigger for SDADC1 and synchronized trigger for SDADC2 (or SDADC3) using HAL_SDADC_SelectRegularTrigger.
3. Select regular channel for SDADC1 and SDADC2 (or SDADC3) using HAL_SDADC_ConfigChannel.
4. Start regular conversion for SDADC2 (or SDADC3) with HAL_SDADC_Start.
5. Start regular conversion for SDADC1 using HAL_SDADC_Start, HAL_SDADC_Start_IT or HAL_SDADC_MultiModeStart_DMA.
6. In polling mode, use HAL_SDADC_PollForConversion to detect the end of regular conversion for SDADC1.
7. In interrupt mode, HAL_SDADC_ConvCpltCallback will be called at the end of regular conversion for SDADC1.
8. Get value of regular conversions using HAL_SDADC_MultiModeGetValue.
9. In DMA mode, HAL_SDADC_ConvHalfCpltCallback and HAL_SDADC_ConvCpltCallback will be called respectively at the half transfer and at the transfer complete for SDADC1.
10. Stop regular conversion using HAL_SDADC_Stop, HAL_SDADC_Stop_IT or HAL_SDADC_MultiModeStop_DMA for SDADC1.
11. Stop regular conversion using HAL_SDADC_Stop for SDADC2 (or SDADC3).

Multi mode injected channels conversions

1. Select type of multimode (SDADC1/SDADC2 or SDADC1/SDADC3) using HAL_SDADC_InjectedMultiModeConfigChannel.
2. Select software or external trigger for SDADC1 and synchronized trigger for SDADC2 (or SDADC3) using HAL_SDADC_SelectInjectedTrigger.
3. Select injected channels for SDADC1 and SDADC2 (or SDADC3) using HAL_SDADC_InjectedConfigChannel.
4. Start injected conversion for SDADC2 (or SDADC3) with HAL_SDADC_InjectedStart.
5. Start injected conversion for SDADC1 using HAL_SDADC_InjectedStart, HAL_SDADC_InjectedStart_IT or HAL_SDADC_InjectedMultiModeStart_DMA.
6. In polling mode, use HAL_SDADC_InjectedPollForConversion to detect the end of injected conversion for SDADC1.
7. In interrupt mode, HAL_SDADC_InjectedConvCpltCallback will be called at the end of injected conversion for SDADC1.
8. Get value of injected conversions using HAL_SDADC_InjectedMultiModeGetValue.
9. In DMA mode, HAL_SDADC_InjectedConvHalfCpltCallback and HAL_SDADC_InjectedConvCpltCallback will be called respectively at the half transfer and at the transfer complete for SDADC1.
10. Stop injected conversion using HAL_SDADC_InjectedStop, HAL_SDADC_InjectedStop_IT or HAL_SDADC_InjecteddMultiModeStop_DMA for SDADC1.
11. Stop injected conversion using HAL_SDADC_InjectedStop for SDADC2 (or SDADC3).

42.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the SDADC.
- De-initialize the SDADC.
- ***HAL_SDADC_Init()***
- ***HAL_SDADC_DeInit()***
- ***HAL_SDADC_MspInit()***

- *HAL_SDADC_MspDeInit()*

42.2.4 Peripheral control functions

This section provides functions allowing to:

- Program one of the three different configurations for channels.
- Associate channel to one of configurations.
- Select regular and injected channels.
- Enable/disable continuous mode for regular and injected conversions.
- Select regular and injected triggers.
- Select and configure injected external trigger.
- Enable/disable delay addition for injected conversions.
- Configure multimode.
- *HAL_SDADC_PrepareChannelConfig()*
- *HAL_SDADC_AssociateChannelConfig()*
- *HAL_SDADC_ConfigChannel()*
- *HAL_SDADC_InjectedConfigChannel()*
- *HAL_SDADC_SelectRegularTrigger()*
- *HAL_SDADC_SelectInjectedTrigger()*
- *HAL_SDADC_SelectInjectedExtTrigger()*
- *HAL_SDADC_SelectInjectedDelay()*
- *HAL_SDADC_MultiModeConfigChannel()*
- *HAL_SDADC_InjectedMultiModeConfigChannel()*

42.2.5 IO operation functions

This section provides functions allowing to:

- Start calibration.
- Poll for the end of calibration.
- Start calibration and enable interrupt.
- Start conversion of regular/injected channel.
- Poll for the end of regular/injected conversion.
- Stop conversion of regular/injected channel.
- Start conversion of regular/injected channel and enable interrupt.
- Stop conversion of regular/injected channel and disable interrupt.
- Start conversion of regular/injected channel and enable DMA transfer.
- Stop conversion of regular/injected channel and disable DMA transfer.
- Start multimode and enable DMA transfer for regular/injected conversion.
- Stop multimode and disable DMA transfer for regular/injected conversion..
- Get result of regular channel conversion.
- Get result of injected channel conversion.
- Get result of multimode conversion.
- Handle SDADC interrupt request.
- Callbacks for calibration and regular/injected conversions.
- *HAL_SDADC_CalibrationStart()*
- *HAL_SDADC_PollForCalibEvent()*
- *HAL_SDADC_CalibrationStart_IT()*
- *HAL_SDADC_Start()*
- *HAL_SDADC_PollForConversion()*
- *HAL_SDADC_Stop()*

- *HAL_SDADC_Start_IT()*
- *HAL_SDADC_Stop_IT()*
- *HAL_SDADC_Start_DMA()*
- *HAL_SDADC_Stop_DMA()*
- *HAL_SDADC_GetValue()*
- *HAL_SDADC_InjectedStart()*
- *HAL_SDADC_PollForInjectedConversion()*
- *HAL_SDADC_InjectedStop()*
- *HAL_SDADC_InjectedStart_IT()*
- *HAL_SDADC_InjectedStop_IT()*
- *HAL_SDADC_InjectedStart_DMA()*
- *HAL_SDADC_InjectedStop_DMA()*
- *HAL_SDADC_InjectedGetValue()*
- *HAL_SDADC_MultiModeStart_DMA()*
- *HAL_SDADC_MultiModeStop_DMA()*
- *HAL_SDADC_MultiModeGetValue()*
- *HAL_SDADC_InjectedMultiModeStart_DMA()*
- *HAL_SDADC_InjectedMultiModeStop_DMA()*
- *HAL_SDADC_InjectedMultiModeGetValue()*
- *HAL_SDADC_IRQHandler()*
- *HAL_SDADC_CalibrationCpltCallback()*
- *HAL_SDADC_ConvHalfCpltCallback()*
- *HAL_SDADC_ConvCpltCallback()*
- *HAL_SDADC_InjectedConvHalfCpltCallback()*
- *HAL_SDADC_InjectedConvCpltCallback()*
- *HAL_SDADC_ErrorCallback()*

42.2.6 ADC Peripheral State functions

This subsection provides functions allowing to

- Get the SDADC state
- Get the SDADC Error
- *HAL_SDADC_GetState()*
- *HAL_SDADC_GetError()*

42.2.7 HAL_SDADC_Init

Function Name	**HAL_StatusTypeDef HAL_SDADC_Init (SDADC_HandleTypeDef * hsdadc)**
Function Description	Initializes the SDADC according to the specified parameters in the SDADC_InitTypeDef structure.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status.
Notes	• If multiple SDADC are used, please configure first SDADC1 to set the common reference voltage.

42.2.8 HAL_SDADC_DeInit

Function Name	**HAL_StatusTypeDef HAL_SDADC_DeInit (SDADC_HandleTypeDef * hsdadc)**
Function Description	De-initializes the SDADC.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status.

42.2.9 HAL_SDADC_MspInit

Function Name	**void HAL_SDADC_MspInit (SDADC_HandleTypeDef * hsdadc)**
Function Description	Initializes the SDADC MSP.
Parameters	• **hsdadc:** SDADC handle
Return values	• None

42.2.10 HAL_SDADC_MspDeInit

Function Name	**void HAL_SDADC_MspDeInit (SDADC_HandleTypeDef * hsdadc)**
Function Description	De-initializes the SDADC MSP.
Parameters	• **hsdadc:** SDADC handle
Return values	• None

42.2.11 HAL_SDADC_PrepareChannelConfig

Function Name	**HAL_StatusTypeDef HAL_SDADC_PrepareChannelConfig (SDADC_HandleTypeDef * hsdadc, uint32_t ConfIndex, SDADC_ConfParamTypeDef * ConfParamStruct)**
Function Description	This function allows the user to set parameters for a configuration.
Parameters	• **hsdadc:** SDADC handle. • **ConfIndex:** Index of configuration to modify. This parameter can be a value of SDADC Configuration Index. • **ConfParamStruct:** Parameters to apply for this configuration.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state (neither calibration nor regular or injected conversion ongoing)

42.2.12 HAL_SDADC_AssociateChannelConfig

Function Name	**HAL_StatusTypeDef HAL_SDADC_AssociateChannelConfig (SDADC_HandleTypeDef * hsdadc, uint32_t Channel, uint32_t ConfIndex)**

Function Description	This function allows the user to associate a channel with one of the available configurations.
Parameters	• **hsdadc:** SDADC handle. • **Channel:** Channel to associate with configuration. This parameter can be a value of SDADC Channel Selection. • **ConfIndex:** Index of configuration to associate with channel. This parameter can be a value of SDADC Configuration Index.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state (neither calibration nor regular or injected conversion ongoing)

42.2.13 HAL_SDADC_ConfigChannel

Function Name	**HAL_StatusTypeDef HAL_SDADC_ConfigChannel (SDADC_HandleTypeDef * hsdadc, uint32_t Channel, uint32_t ContinuousMode)**
Function Description	This function allows to select channel for regular conversion and to enable/disable continuous mode for regular conversion.
Parameters	• **hsdadc:** SDADC handle. • **Channel:** Channel for regular conversion. This parameter can be a value of SDADC Channel Selection. • **ContinuousMode:** Enable/disable continuous mode for regular conversion. This parameter can be a value of SDADC Continuous Mode.
Return values	• HAL status

42.2.14 HAL_SDADC_InjectedConfigChannel

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedConfigChannel (SDADC_HandleTypeDef * hsdadc, uint32_t Channel, uint32_t ContinuousMode)**
Function Description	This function allows to select channels for injected conversion and to enable/disable continuous mode for injected conversion.
Parameters	• **hsdadc:** SDADC handle. • **Channel:** Channels for injected conversion. This parameter can be a values combination of SDADC Channel Selection. • **ContinuousMode:** Enable/disable continuous mode for injected conversion. This parameter can be a value of SDADC Continuous Mode.
Return values	• HAL status

42.2.15 HAL_SDADC_SelectRegularTrigger

Function Name	**HAL_StatusTypeDef HAL_SDADC_SelectRegularTrigger (SDADC_HandleTypeDef * hsdadc, uint32_t Trigger)**
Function Description	This function allows to select trigger for regular conversions.
Parameters	• **hsdadc:** SDADC handle. • **Trigger:** Trigger for regular conversions. This parameter can be one of the following value : SDADC_SOFTWARE_TRIGGER : Software trigger. SDADC_SYNCHRONOUS_TRIGGER : Synchronous with SDADC1 (only for SDADC2 and SDADC3).
Return values	• HAL status
Notes	• This function should not be called if regular conversion is ongoing.

42.2.16 HAL_SDADC_SelectInjectedTrigger

Function Name	**HAL_StatusTypeDef HAL_SDADC_SelectInjectedTrigger (SDADC_HandleTypeDef * hsdadc, uint32_t Trigger)**
Function Description	This function allows to select trigger for injected conversions.
Parameters	• **hsdadc:** SDADC handle. • **Trigger:** Trigger for injected conversions. This parameter can be one of the following value : SDADC_SOFTWARE_TRIGGER : Software trigger. SDADC_SYNCHRONOUS_TRIGGER : Synchronous with SDADC1 (only for SDADC2 and SDADC3). SDADC_EXTERNAL_TRIGGER : External trigger.
Return values	• HAL status
Notes	• This function should not be called if injected conversion is ongoing.

42.2.17 HAL_SDADC_SelectInjectedExtTrigger

Function Name	**HAL_StatusTypeDef HAL_SDADC_SelectInjectedExtTrigger (SDADC_HandleTypeDef * hsdadc, uint32_t InjectedExtTrigger, uint32_t ExtTriggerEdge)**
Function Description	This function allows to select and configure injected external trigger.
Parameters	• **hsdadc:** SDADC handle. • **InjectedExtTrigger:** External trigger for injected conversions. This parameter can be a value of SDADC Injected External Trigger. • **ExtTriggerEdge:** Edge of external injected trigger. This parameter can be a value of SDADC External Trigger Edge.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state (neither calibration nor regular or injected

conversion ongoing)

42.2.18 HAL_SDADC_SelectInjectedDelay

Function Name	**HAL_StatusTypeDef HAL_SDADC_SelectInjectedDelay (SDADC_HandleTypeDef * hsdadc, uint32_t InjectedDelay)**
Function Description	This function allows to enable/disable delay addition for injected conversions.
Parameters	• **hsdadc:** SDADC handle. • **InjectedDelay:** Enable/disable delay for injected conversions. This parameter can be a value of SDADC Injected Conversion Delay.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state (neither calibration nor regular or injected conversion ongoing)

42.2.19 HAL_SDADC_MultiModeConfigChannel

Function Name	**HAL_StatusTypeDef HAL_SDADC_MultiModeConfigChannel (SDADC_HandleTypeDef * hsdadc, uint32_t MultimodeType)**
Function Description	This function allows to configure multimode for regular conversions.
Parameters	• **hsdadc:** SDADC handle. • **MultimodeType:** Type of multimode for regular conversions. This parameter can be a value of SDADC Multimode Type.
Return values	• HAL status
Notes	• This function should not be called if regular conversion is ongoing and should be could only for SDADC1.

42.2.20 HAL_SDADC_InjectedMultiModeConfigChannel

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedMultiModeConfigChannel (SDADC_HandleTypeDef * hsdadc, uint32_t MultimodeType)**
Function Description	This function allows to configure multimode for injected conversions.
Parameters	• **hsdadc:** SDADC handle. • **MultimodeType:** Type of multimode for injected conversions. This parameter can be a value of SDADC Multimode Type.
Return values	• HAL status
Notes	• This function should not be called if injected conversion is ongoing and should be could only for SDADC1.

42.2.21 HAL_SDADC_CalibrationStart

Function Name	**HAL_StatusTypeDef HAL_SDADC_CalibrationStart (SDADC_HandleTypeDef * hsdadc, uint32_t CalibrationSequence)**
Function Description	This function allows to start calibration in polling mode.
Parameters	• **hsdadc:** SDADC handle. • **CalibrationSequence:** Calibration sequence. This parameter can be a value of SDADC Calibration Sequence.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state (neither calibration nor regular or injected conversion ongoing).

42.2.22 HAL_SDADC_PollForCalibEvent

Function Name	**HAL_StatusTypeDef HAL_SDADC_PollForCalibEvent (SDADC_HandleTypeDef * hsdadc, uint32_t Timeout)**
Function Description	This function allows to poll for the end of calibration.
Parameters	• **hsdadc:** SDADC handle. • **Timeout:** Timeout value in milliseconds.
Return values	• HAL status
Notes	• This function should be called only if calibration is ongoing.

42.2.23 HAL_SDADC_CalibrationStart_IT

Function Name	**HAL_StatusTypeDef HAL_SDADC_CalibrationStart_IT (SDADC_HandleTypeDef * hsdadc, uint32_t CalibrationSequence)**
Function Description	This function allows to start calibration in interrupt mode.
Parameters	• **hsdadc:** SDADC handle. • **CalibrationSequence:** Calibration sequence. This parameter can be a value of SDADC Calibration Sequence.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state (neither calibration nor regular or injected conversion ongoing).

42.2.24 HAL_SDADC_Start

Function Name	**HAL_StatusTypeDef HAL_SDADC_Start (SDADC_HandleTypeDef * hsdadc)**

Function Description	This function allows to start regular conversion in polling mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state or if injected conversion is ongoing.

42.2.25 HAL_SDADC_PollForConversion

Function Name	**HAL_StatusTypeDef HAL_SDADC_PollForConversion (SDADC_HandleTypeDef * hsdadc, uint32_t Timeout)**
Function Description	This function allows to poll for the end of regular conversion.
Parameters	• **hsdadc:** SDADC handle. • **Timeout:** Timeout value in milliseconds.
Return values	• HAL status
Notes	• This function should be called only if regular conversion is ongoing.

42.2.26 HAL_SDADC_Stop

Function Name	**HAL_StatusTypeDef HAL_SDADC_Stop (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to stop regular conversion in polling mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only if regular conversion is ongoing.

42.2.27 HAL_SDADC_Start_IT

Function Name	**HAL_StatusTypeDef HAL_SDADC_Start_IT (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to start regular conversion in interrupt mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state or if injected conversion is ongoing.

42.2.28 HAL_SDADC_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_SDADC_Stop_IT**

	(SDADC_HandleTypeDef * hsdadc)
Function Description	This function allows to stop regular conversion in interrupt mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only if regular conversion is ongoing.

42.2.29 HAL_SDADC_Start_DMA

Function Name	**HAL_StatusTypeDef HAL_SDADC_Start_DMA (SDADC_HandleTypeDef * hsdadc, uint32_t * pData, uint32_t Length)**
Function Description	This function allows to start regular conversion in DMA mode.
Parameters	• **hsdadc:** SDADC handle. • **pData:** The destination buffer address. • **Length:** The length of data to be transferred from SDADC peripheral to memory.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state or if injected conversion is ongoing.

42.2.30 HAL_SDADC_Stop_DMA

Function Name	**HAL_StatusTypeDef HAL_SDADC_Stop_DMA (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to stop regular conversion in DMA mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only if regular conversion is ongoing.

42.2.31 HAL_SDADC_GetValue

Function Name	**uint32_t HAL_SDADC_GetValue (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to get regular conversion value.
Parameters	• **hsdadc:** SDADC handle.
Return values	• Regular conversion value

42.2.32 HAL_SDADC_InjectedStart

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedStart (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to start injected conversion in polling mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state or if regular conversion is ongoing.

42.2.33 HAL_SDADC_PollForInjectedConversion

Function Name	**HAL_StatusTypeDef HAL_SDADC_PollForInjectedConversion (SDADC_HandleTypeDef * hsdadc, uint32_t Timeout)**
Function Description	This function allows to poll for the end of injected conversion.
Parameters	• **hsdadc:** SDADC handle. • **Timeout:** Timeout value in milliseconds.
Return values	• HAL status
Notes	• This function should be called only if injected conversion is ongoing.

42.2.34 HAL_SDADC_InjectedStop

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedStop (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to stop injected conversion in polling mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only if injected conversion is ongoing.

42.2.35 HAL_SDADC_InjectedStart_IT

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedStart_IT (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to start injected conversion in interrupt mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state or if regular conversion is ongoing.

42.2.36 HAL_SDADC_InjectedStop_IT

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedStop_IT (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to stop injected conversion in interrupt mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only if injected conversion is ongoing.

42.2.37 HAL_SDADC_InjectedStart_DMA

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedStart_DMA (SDADC_HandleTypeDef * hsdadc, uint32_t * pData, uint32_t Length)**
Function Description	This function allows to start injected conversion in DMA mode.
Parameters	• **hsdadc:** SDADC handle. • **pData:** The destination buffer address. • **Length:** The length of data to be transferred from SDADC peripheral to memory.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state or if regular conversion is ongoing.

42.2.38 HAL_SDADC_InjectedStop_DMA

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedStop_DMA (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to stop injected conversion in DMA mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only if injected conversion is ongoing.

42.2.39 HAL_SDADC_InjectedGetValue

Function Name	**uint32_t HAL_SDADC_InjectedGetValue (SDADC_HandleTypeDef * hsdadc, uint32_t * Channel)**
Function Description	This function allows to get injected conversion value.
Parameters	• **hsdadc:** SDADC handle. • **Channel:** Corresponding channel of injected conversion.

Return values	•	Injected conversion value

42.2.40 HAL_SDADC_MultiModeStart_DMA

Function Name	**HAL_StatusTypeDef HAL_SDADC_MultiModeStart_DMA (SDADC_HandleTypeDef * hsdadc, uint32_t * pData, uint32_t Length)**
Function Description	This function allows to start multimode regular conversions in DMA mode.
Parameters	• **hsdadc:** SDADC handle. • **pData:** The destination buffer address. • **Length:** The length of data to be transferred from SDADC peripheral to memory.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state or if injected conversion is ongoing.

42.2.41 HAL_SDADC_MultiModeStop_DMA

Function Name	**HAL_StatusTypeDef HAL_SDADC_MultiModeStop_DMA (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to stop multimode regular conversions in DMA mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only if regular conversion is ongoing.

42.2.42 HAL_SDADC_MultiModeGetValue

Function Name	**uint32_t HAL_SDADC_MultiModeGetValue (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to get multimode regular conversion value.
Parameters	• **hsdadc:** SDADC handle.
Return values	• Multimode regular conversion value

42.2.43 HAL_SDADC_InjectedMultiModeStart_DMA

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedMultiModeStart_DMA (SDADC_HandleTypeDef * hsdadc, uint32_t * pData, uint32_t Length)**

Function Description	This function allows to start multimode injected conversions in DMA mode.
Parameters	• **hsdadc:** SDADC handle. • **pData:** The destination buffer address. • **Length:** The length of data to be transferred from SDADC peripheral to memory.
Return values	• HAL status
Notes	• This function should be called only when SDADC instance is in idle state or if regular conversion is ongoing.

42.2.44 HAL_SDADC_InjectedMultiModeStop_DMA

Function Name	**HAL_StatusTypeDef HAL_SDADC_InjectedMultiModeStop_DMA (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to stop multimode injected conversions in DMA mode.
Parameters	• **hsdadc:** SDADC handle.
Return values	• HAL status
Notes	• This function should be called only if injected conversion is ongoing.

42.2.45 HAL_SDADC_InjectedMultiModeGetValue

Function Name	**uint32_t HAL_SDADC_InjectedMultiModeGetValue (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to get multimode injected conversion value.
Parameters	• **hsdadc:** SDADC handle.
Return values	• Multimode injected conversion value

42.2.46 HAL_SDADC_IRQHandler

Function Name	**void HAL_SDADC_IRQHandler (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function handles the SDADC interrupts.
Parameters	• **hsdadc:** SDADC handle.
Return values	• None

42.2.47 HAL_SDADC_CalibrationCpltCallback

Function Name	**void HAL_SDADC_CalibrationCpltCallback**

	(SDADC_HandleTypeDef * hsdadc)
Function Description	Calibration complete callback.
Parameters	• **hsdadc:** SDADC handle.
Return values	• None

42.2.48 HAL_SDADC_ConvHalfCpltCallback

Function Name	**void HAL_SDADC_ConvHalfCpltCallback (SDADC_HandleTypeDef * hsdadc)**
Function Description	Half regular conversion complete callback.
Parameters	• **hsdadc:** SDADC handle.
Return values	• None

42.2.49 HAL_SDADC_ConvCpltCallback

Function Name	**void HAL_SDADC_ConvCpltCallback (SDADC_HandleTypeDef * hsdadc)**
Function Description	Regular conversion complete callback.
Parameters	• **hsdadc:** SDADC handle.
Return values	• None
Notes	• In interrupt mode, user has to read conversion value in this function using HAL_SDADC_GetValue or HAL_SDADC_MultiModeGetValue.

42.2.50 HAL_SDADC_InjectedConvHalfCpltCallback

Function Name	**void HAL_SDADC_InjectedConvHalfCpltCallback (SDADC_HandleTypeDef * hsdadc)**
Function Description	Half injected conversion complete callback.
Parameters	• **hsdadc:** SDADC handle.
Return values	• None

42.2.51 HAL_SDADC_InjectedConvCpltCallback

Function Name	**void HAL_SDADC_InjectedConvCpltCallback (SDADC_HandleTypeDef * hsdadc)**
Function Description	Injected conversion complete callback.
Parameters	• **hsdadc:** SDADC handle.
Return values	• None

Notes	• In interrupt mode, user has to read conversion value in this function using HAL_SDADC_InjectedGetValue or HAL_SDADC_InjectedMultiModeGetValue.

42.2.52 HAL_SDADC_ErrorCallback

Function Name	**void HAL_SDADC_ErrorCallback (SDADC_HandleTypeDef * hsdadc)**
Function Description	Error callback.
Parameters	• **hsdadc:** SDADC handle.
Return values	• None

42.2.53 HAL_SDADC_GetState

Function Name	**HAL_SDADC_StateTypeDef HAL_SDADC_GetState (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to get the current SDADC state.
Parameters	• **hsdadc:** SDADC handle.
Return values	• SDADC state.

42.2.54 HAL_SDADC_GetError

Function Name	**uint32_t HAL_SDADC_GetError (SDADC_HandleTypeDef * hsdadc)**
Function Description	This function allows to get the current SDADC error code.
Parameters	• **hsdadc:** SDADC handle.
Return values	• SDADC error code.

42.3 SDADC Firmware driver defines

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

42.3.1 SDADC

SDADC

SDADC Calibration Sequence

SDADC_CALIBRATION_SEQ_1	One calibration sequence to calculate offset of conf0 (OFFSET0[11:0])
SDADC_CALIBRATION_SEQ_2	Two calibration sequences to calculate offset of conf0 and conf1 (OFFSET0[11:0] and OFFSET1[11:0])
SDADC_CALIBRATION_SEQ_3	Three calibration sequences to calculate offset of conf0, conf1 and conf2 (OFFSET0[11:0], OFFSET1[11:0], and OFFSET2[11:0])

SDADC Channel Selection

SDADC_CHANNEL_0

SDADC_CHANNEL_1

SDADC_CHANNEL_2

SDADC_CHANNEL_3

SDADC_CHANNEL_4

SDADC_CHANNEL_5

SDADC_CHANNEL_6

SDADC_CHANNEL_7

SDADC_CHANNEL_8

SDADC Common Mode

SDADC_COMMON_MODE_VSSA	Select SDADC VSSA as common mode
SDADC_COMMON_MODE_VDDA_2	Select SDADC VDDA/2 as common mode
SDADC_COMMON_MODE_VDDA	Select SDADC VDDA as common mode

SDADC Configuration Index

SDADC_CONF_INDEX_0	Configuration 0 Register selected
SDADC_CONF_INDEX_1	Configuration 1 Register selected
SDADC_CONF_INDEX_2	Configuration 2 Register selected

SDADC Continuous Mode

SDADC_CONTINUOUS_CONV_OFF	Conversion are not continuous
SDADC_CONTINUOUS_CONV_ON	Conversion are continuous

SDADC Error Code

SDADC_ERROR_NONE	No error
SDADC_ERROR_REGULAR_OVERRUN	Overrun occurs during regular conversion
SDADC_ERROR_INJECTED_OVERRUN	Overrun occurs during injected conversion
SDADC_ERROR_DMA	DMA error occurs

SDADC Exported Macros

__HAL_SDADC_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:
  – SDADC_IT_EOCAL: End of calibration interrupt enable
  – SDADC_IT_JEOC: Injected end of conversion interrupt enable
  – SDADC_IT_JOVR: Injected data

overrun interrupt enable
- – SDADC_IT_REOC: Regular end of conversion interrupt enable
- – SDADC_IT_ROVR: Regular data overrun interrupt enable

Return value:

- None:

__HAL_SDADC_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:
  - – SDADC_IT_EOCAL: End of calibration interrupt enable
  - – SDADC_IT_JEOC: Injected end of conversion interrupt enable
  - – SDADC_IT_JOVR: Injected data overrun interrupt enable
  - – SDADC_IT_REOC: Regular end of conversion interrupt enable
  - – SDADC_IT_ROVR: Regular data overrun interrupt enable

Return value:

- None:

__HAL_SDADC_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:
  - – SDADC_IT_EOCAL: End of calibration interrupt enable
  - – SDADC_IT_JEOC: Injected end of conversion interrupt enable
  - – SDADC_IT_JOVR: Injected data overrun interrupt enable
  - – SDADC_IT_REOC: Regular end of conversion interrupt enable
  - – SDADC_IT_ROVR: Regular data overrun interrupt enable

Return value:

- State: of interruption (SET or RESET)

__HAL_SDADC_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:
  - SDADC_FLAG_EOCAL: End of calibration flag
  - SDADC_FLAG_JEOC: End of injected conversion flag
  - SDADC_FLAG_JOVR: Injected conversion overrun flag
  - SDADC_FLAG_REOC: End of regular conversion flag
  - SDADC_FLAG_ROVR: Regular conversion overrun flag

Return value:

- None:

__HAL_SDADC_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:
  - SDADC_FLAG_EOCAL: End of calibration flag
  - SDADC_FLAG_JEOC: End of injected conversion flag
  - SDADC_FLAG_JOVR: Injected conversion overrun flag
  - SDADC_FLAG_REOC: End of regular conversion flag
  - SDADC_FLAG_ROVR: Regular conversion overrun flag

Return value:

- None:

__HAL_SDADC_RESET_HANDLE_STATE

Description:

- Reset SDADC handle state.

Parameters:

- __HANDLE__: SDADC handle.

Return value:

- None:

SDADC External Trigger Edge

SDADC_EXT_TRIG_RISING_EDGE	External rising edge
SDADC_EXT_TRIG_FALLING_EDGE	External falling edge
SDADC_EXT_TRIG_BOTH_EDGES	External rising and falling edges

SDADC Fast Conversion Mode

SDADC_FAST_CONV_DISABLE

SDADC_FAST_CONV_ENABLE

SDADC flags definition

SDADC_FLAG_EOCAL	End of calibration flag
SDADC_FLAG_JEOC	End of injected conversion flag
SDADC_FLAG_JOVR	Injected conversion overrun flag
SDADC_FLAG_REOC	End of regular conversion flag
SDADC_FLAG_ROVR	Regular conversion overrun flag

SDADC Gain

SDADC_GAIN_1	Gain equal to 1
SDADC_GAIN_2	Gain equal to 2
SDADC_GAIN_4	Gain equal to 4
SDADC_GAIN_8	Gain equal to 8
SDADC_GAIN_16	Gain equal to 16
SDADC_GAIN_32	Gain equal to 32
SDADC_GAIN_1_2	Gain equal to 1/2

SDADC Idle Low Power Mode

SDADC_LOWPOWER_NONE

SDADC_LOWPOWER_POWERDOWN

SDADC_LOWPOWER_STANDBY

SDADC Injected Conversion Delay

SDADC_INJECTED_DELAY_NONE	No delay on injected conversion
SDADC_INJECTED_DELAY	Delay on injected conversion

SDADC Injected External Trigger

SDADC_EXT_TRIG_TIM13_CC1	Trigger source for SDADC1
SDADC_EXT_TRIG_TIM14_CC1	Trigger source for SDADC1
SDADC_EXT_TRIG_TIM16_CC1	Trigger source for SDADC3
SDADC_EXT_TRIG_TIM17_CC1	Trigger source for SDADC2
SDADC_EXT_TRIG_TIM12_CC1	Trigger source for SDADC2
SDADC_EXT_TRIG_TIM12_CC2	Trigger source for SDADC3

SDADC_EXT_TRIG_TIM15_CC2	Trigger source for SDADC1
SDADC_EXT_TRIG_TIM2_CC3	Trigger source for SDADC2
SDADC_EXT_TRIG_TIM2_CC4	Trigger source for SDADC3
SDADC_EXT_TRIG_TIM3_CC1	Trigger source for SDADC1
SDADC_EXT_TRIG_TIM3_CC2	Trigger source for SDADC2
SDADC_EXT_TRIG_TIM3_CC3	Trigger source for SDADC3
SDADC_EXT_TRIG_TIM4_CC1	Trigger source for SDADC1
SDADC_EXT_TRIG_TIM4_CC2	Trigger source for SDADC2
SDADC_EXT_TRIG_TIM4_CC3	Trigger source for SDADC3
SDADC_EXT_TRIG_TIM19_CC2	Trigger source for SDADC1
SDADC_EXT_TRIG_TIM19_CC3	Trigger source for SDADC2
SDADC_EXT_TRIG_TIM19_CC4	Trigger source for SDADC3
SDADC_EXT_TRIG_EXTI11	Trigger source for SDADC1, SDADC2 and SDADC3
SDADC_EXT_TRIG_EXTI15	Trigger source for SDADC1, SDADC2 and SDADC3

SDADC Input Mode

SDADC_INPUT_MODE_DIFF	Conversions are executed in differential mode
SDADC_INPUT_MODE_SE_OFFSET	Conversions are executed in single ended offset mode
SDADC_INPUT_MODE_SE_ZERO_REFERENCE	Conversions are executed in single ended zero-volt reference mode

SDADC interrupts definition

SDADC_IT_EOCAL	End of calibration interrupt enable
SDADC_IT_JEOC	Injected end of conversion interrupt enable
SDADC_IT_JOVR	Injected data overrun interrupt enable
SDADC_IT_REOC	Regular end of conversion interrupt enable
SDADC_IT_ROVR	Regular data overrun interrupt enable

SDADC Multimode Type

SDADC_MULTIMODE_SDADC1_SDADC2	Get conversion values for SDADC1 and SDADC2
SDADC_MULTIMODE_SDADC1_SDADC3	Get conversion values for SDADC1 and SDADC3

SDADC Private Define

SDADC_TIMEOUT

SDADC_CONFREG_OFFSET

SDADC_JDATAR_CH_OFFSET

SDADC_MSB_MASK

SDADC_LSB_MASK

SDADC Private Macros

IS_SDADC_LOWPOWER_MODE

IS_SDADC_FAST_CONV_MODE

IS_SDADC_SLOW_CLOCK_MODE

IS_SDADC_VREF

IS_SDADC_CONF_INDEX

IS_SDADC_INPUT_MODE

IS_SDADC_GAIN

IS_SDADC_COMMON_MODE

IS_SDADC_OFFSET_VALUE

IS_SDADC_REGULAR_CHANNEL

IS_SDADC_INJECTED_CHANNEL

IS_SDADC_CALIB_SEQUENCE

IS_SDADC_CONTINUOUS_MODE

IS_SDADC_REGULAR_TRIGGER

IS_SDADC_INJECTED_TRIGGER

IS_SDADC_EXT_INJEC_TRIG

IS_SDADC_EXT_TRIG_EDGE

IS_SDADC_INJECTED_DELAY

IS_SDADC_MULTIMODE_TYPE

SDADC Reference Voltage

SDADC_VREF_EXT	The reference voltage is forced externally using VREF pin
SDADC_VREF_VREFINT1	The reference voltage is forced internally to 1.22V VREFINT
SDADC_VREF_VREFINT2	The reference voltage is forced internally to 1.8V VREFINT
SDADC_VREF_VDDA	The reference voltage is forced internally to VDDA

SDADC Slow Clock Mode

SDADC_SLOW_CLOCK_DISABLE

SDADC_SLOW_CLOCK_ENABLE

SDADC Trigger

SDADC_SOFTWARE_TRIGGER	Software trigger
SDADC_SYNCHRONOUS_TRIGGER	Synchronous with SDADC1 (only for SDADC2 and SDADC3)
SDADC_EXTERNAL_TRIGGER	External trigger

43 HAL SMARTCARD Generic Driver

43.1 SMARTCARD Firmware driver registers structures

43.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32f3xx_hal_smartcard.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint16_t Parity***
- ***uint16_t Mode***
- ***uint16_t CLKPolarity***
- ***uint16_t CLKPhase***
- ***uint16_t CLKLastBit***
- ***uint16_t OneBitSampling***
- ***uint8_t Prescaler***
- ***uint8_t GuardTime***
- ***uint16_t NACKEnable***
- ***uint32_t TimeOutEnable***
- ***uint32_t TimeOutValue***
- ***uint8_t BlockLength***
- ***uint8_t AutoRetryCount***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***
  Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hsmartcard->Init.BaudRate)))
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter ***SMARTCARD_Word_Length*** can only be set to 9 (8 data + 1 parity bits).
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***
  Specifies the number of stop bits ***SMARTCARD_Stop_Bits***. Only 1.5 stop bits are authorized in SmartCard mode.
- ***uint16_t SMARTCARD_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***SMARTCARD_Parity***
  Note: The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- ***uint16_t SMARTCARD_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***

- ***uint16_t SMARTCARD_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD_Clock_Phase***
- ***uint16_t SMARTCARD_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD_Last_Bit***
- ***uint16_t SMARTCARD_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***SMARTCARD_OneBit_Sampling***.
- ***uint8_t SMARTCARD_InitTypeDef::Prescaler***
  Specifies the SmartCard Prescaler
- ***uint8_t SMARTCARD_InitTypeDef::GuardTime***
  Specifies the SmartCard Guard Time
- ***uint16_t SMARTCARD_InitTypeDef::NACKEnable***
  Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of ***SMARTCARD_NACK_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutEnable***
  Specifies whether the receiver timeout is enabled. This parameter can be a value of ***SMARTCARD_Timeout_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutValue***
  Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint8_t SMARTCARD_InitTypeDef::BlockLength***
  Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint8_t SMARTCARD_InitTypeDef::AutoRetryCount***
  Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)

43.1.2 SMARTCARD_AdvFeatureInitTypeDef

SMARTCARD_AdvFeatureInitTypeDef is defined in the stm32f3xx_hal_smartcard.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t MSBFirst***

Field Documentation

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced SMARTCARD features is initialized. Several advanced

features may be initialized at the same time. This parameter can be a value of ***SMARTCARD_Advanced_Features_Initialization_Type***

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Tx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Rx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***SMARTCARD_Data_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***SMARTCARD_Rx_Tx_Swap***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***SMARTCARD_Overrun_Disable***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***SMARTCARD_DMA_Disable_on_Rx_Error***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***SMARTCARD_MSB_First***

43.1.3 SMARTCARD_HandleTypeDef

SMARTCARD_HandleTypeDef is defined in the stm32f3xx_hal_smartcard.h

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***SMARTCARD_AdvFeatureInitTypeDef AdvancedInit***
- ***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***
- ***HAL_SMARTCARD_StateTypeDef State***
- ***HAL_SMARTCARD_ErrorTypeDef ErrorCode***

Field Documentation

- ***USART_TypeDef* SMARTCARD_HandleTypeDef::Instance***
  USART registers base address
- ***SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init***
  SmartCard communication parameters

- ***SMARTCARD_AdvFeatureInitTypeDef SMARTCARD_HandleTypeDef::AdvancedInit***
  SmartCard advanced features initialization parameters
- ***uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr***
  Pointer to SmartCard Tx transfer Buffer
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferSize***
  SmartCard Tx Transfer size
- ***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
- ***HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State***
  SmartCard communication state
- ***HAL_SMARTCARD_ErrorTypeDef SMARTCARD_HandleTypeDef::ErrorCode***
  SmartCard Error code

43.2 SMARTCARD Firmware driver API description

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

43.2.1 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:
   - Enable the USARTx interface clock.
   - SMARTCARD pins configuration:
     - Enable the clock for the SMARTCARD GPIOs.
     - Configure these SMARTCARD pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.

   - Configure the DMA Tx/Rx channel.
   - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
   - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard Init structure.
4. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.
5. Initialize the SMARTCARD associated USART 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() 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.
6. 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

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

43.2.2 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:
  – Baud Rate
  – Parity: parity should be enabled, Frame Length is fixed to 8 bits plus parity: the USART frame format is given in the tables below.
  – Receiver/transmitter modes
  – Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  – Prescaler value
  – Guard bit time
  – NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  – TX and/or RX pin level inversion
  – data logical level inversion
  – RX and TX pins swap
  – RX overrun detection disabling
  – DMA disabling on RX error
  – MSB first on communication line

– Time out enabling (and if activated, timeout value)
– Block length
– Auto-retry counter

Table 26: USART frame formats (1 M bit)

M bit	PCE bit	USART frame
1	1	| SB | 8 bit data | PB | STB |

Table 27: USART frame formats (2 M bits)

M1, M0 bits	PCE bit	USART frame
01	1	| SB | 8 bit data | PB | STB |

The HAL_SMARTCARD_Init() API follow respectively the USART (a)synchronous configuration procedures (details for the procedures are available in reference manual).

- ***HAL_SMARTCARD_Init()***
- ***HAL_SMARTCARD_DeInit()***
- ***HAL_SMARTCARD_MspInit()***
- ***HAL_SMARTCARD_MspDeInit()***

43.2.3 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.
   – Non Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated 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 respectivelly 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 API's are :
   – HAL_SMARTCARD_Transmit()
   – HAL_SMARTCARD_Receive()
3. Non Blocking mode API's 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()

- ***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()***

43.2.4 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()***

43.2.5 HAL_SMARTCARD_Init

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	Initializes the SMARTCARD mode according to the specified parameters in the SMARTCARD_InitTypeDef and creates the associated handle .
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• HAL status

43.2.6 HAL_SMARTCARD_DeInit

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	DeInitializes the SMARTCARD peripheral.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• HAL status

43.2.7 HAL_SMARTCARD_MspInit

Function Name	**void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	SMARTCARD MSP Init.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• None

43.2.8 HAL_SMARTCARD_MspDeInit

Function Name	**void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	SMARTCARD MSP DeInit.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• None

43.2.9 HAL_SMARTCARD_Transmit

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Send an amount of data in blocking mode.
Parameters	• **hsmartcard:** SMARTCARD handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent • **Timeout:** Timeout duration
Return values	• HAL status

43.2.10 HAL_SMARTCARD_Receive

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Receive an amount of data in blocking mode.
Parameters	• **hsmartcard:** SMARTCARD handle • **pData:** pointer to data buffer • **Size:** amount of data to be received • **Timeout:** Timeout duration
Return values	• HAL status

43.2.11 HAL_SMARTCARD_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)**
Function Description	Send an amount of data in interrupt mode.
Parameters	• **hsmartcard:** SMARTCARD handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

43.2.12 HAL_SMARTCARD_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)**
Function Description	Receive an amount of data in interrupt mode.
Parameters	• **hsmartcard:** SMARTCARD handle • **pData:** pointer to data buffer • **Size:** amount of data to be received
Return values	• HAL status

43.2.13 HAL_SMARTCARD_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)**
Function Description	Send an amount of data in DMA mode.
Parameters	• **hsmartcard:** SMARTCARD handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

43.2.14 HAL_SMARTCARD_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)**
Function Description	Receive an amount of data in DMA mode.
Parameters	• **hsmartcard:** SMARTCARD handle • **pData:** pointer to data buffer • **Size:** amount of data to be received

Return values	•	HAL status
Notes	•	The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

43.2.15 HAL_SMARTCARD_IRQHandler

Function Name	**void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	SMARTCARD interrupt requests handling.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• None

43.2.16 HAL_SMARTCARD_TxCpltCallback

Function Name	**void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	Tx Transfer completed callbacks.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• None

43.2.17 HAL_SMARTCARD_RxCpltCallback

Function Name	**void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	Rx Transfer completed callbacks.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• None

43.2.18 HAL_SMARTCARD_ErrorCallback

Function Name	**void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	SMARTCARD error callbacks.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• None

43.2.19 HAL_SMARTCARD_GetState

Function Name	**HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef ***

	hsmartcard)
Function Description	return the SMARTCARD state
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• HAL state

43.2.20 HAL_SMARTCARD_GetError

Function Name	**uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	Return the SMARTCARD error code.
Parameters	• **hsmartcard:** pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD.
Return values	• SMARTCARD Error Code

43.3 SMARTCARD Firmware driver defines

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

43.3.1 SMARTCARD

SMARTCARD

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT

SMARTCARD_ADVFEATURE_TXINVERT_INIT

SMARTCARD_ADVFEATURE_RXINVERT_INIT

SMARTCARD_ADVFEATURE_DATAINVERT_INIT

SMARTCARD_ADVFEATURE_SWAP_INIT

SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

IS_SMARTCARD_ADVFEATURE_INIT

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD_PHASE_2EDGE

IS_SMARTCARD_PHASE

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD_POLARITY_HIGH

IS_SMARTCARD_POLARITY

SMARTCARD auto retry counter LSB position in CR3 register

SMARTCARD_CR3_SCARCNT_LSB_POS

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE

SMARTCARD_ADVFEATURE_DATAINV_ENABLE

IS_SMARTCARD_ADVFEATURE_DATAINV

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR

SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR

IS_SMARTCARD_ADVFEATURE_DMAONRXERROR

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle state.

Parameters:

- __HANDLE__: SMARTCARD handle.

Return value:

- None:

__HAL_SMARTCARD_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD_FLAG_REACK: Receive enable ackowledge flag
  - SMARTCARD_FLAG_TEACK: Transmit enable ackowledge flag
  - SMARTCARD_FLAG_BUSY: Busy flag
  - SMARTCARD_FLAG_EOBF: End of block flag
  - SMARTCARD_FLAG_RTOF: Receiver timeout flag
  - SMARTCARD_FLAG_TXE: Transmit data register empty flag
  - SMARTCARD_FLAG_TC: Transmission Complete flag
  - SMARTCARD_FLAG_RXNE: Receive data register not empty flag
  - SMARTCARD_FLAG_ORE: OverRun Error flag

- SMARTCARD_FLAG_NE: Noise Error flag
- SMARTCARD_FLAG_FE: Framing Error flag
- SMARTCARD_FLAG_PE: Parity Error flag

Return value:

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

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enables the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOBF: End Of Block interrupt
  - SMARTCARD_IT_RTOF: Receive TimeOut interrupt
  - 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_PE: Parity Error interrupt
  - SMARTCARD_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None:

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disables the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:

- SMARTCARD_IT_EOBF: End Of Block interrupt
- SMARTCARD_IT_RTOF: Receive TimeOut interrupt
- 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_PE: Parity Error interrupt
- SMARTCARD_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None:

__HAL_SMARTCARD_GET_IT

Description:

- Checks whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- __IT__: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOBF: End Of Block interrupt
  - SMARTCARD_IT_RTOF: Receive TimeOut interrupt
  - 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_ORE: OverRun Error interrupt
  - SMARTCARD_IT_NE: Noise Error interrupt
  - SMARTCARD_IT_FE: Framing Error interrupt
  - SMARTCARD_IT_PE: Parity Error interrupt

Return value:

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

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Checks whether the specified SmartCard interrupt interrupt source is enabled.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- __IT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOBF: End Of Block interrupt
  - SMARTCARD_IT_RTOF: Receive TimeOut interrupt
  - 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_ORE: OverRun Error interrupt
  - SMARTCARD_IT_NE: Noise Error interrupt
  - SMARTCARD_IT_FE: Framing Error interrupt
  - SMARTCARD_IT_PE: Parity Error interrupt

Return value:

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

__HAL_SMARTCARD_CLEAR_IT

Description:

- Clears the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - USART_CLEAR_PEF: Parity Error Clear Flag
  - USART_CLEAR_FEF: Framing Error Clear Flag
  - USART_CLEAR_NEF: Noise detected

Clear Flag
- USART_CLEAR_OREF: OverRun Error Clear Flag
- USART_CLEAR_TCF: Transmission Complete Clear Flag
- USART_CLEAR_RTOF: Receiver Time Out Clear Flag
- USART_CLEAR_EOBF: End Of Block Clear Flag

Return value:

- None:

__HAL_SMARTCARD_SEND_REQ

Description:

- Set a specific SMARTCARD request flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be USARTx where x: 1, 2 or 3 to select the USART peripheral.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD_RXDATA_FLUSH_REQUEST: Receive Data flush Request
  - SMARTCARD_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None:

__HAL_SMARTCARD_ENABLE

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be UARTx where x: 1, 2, 3 to select the USART peripheral

Return value:

- None:

__HAL_SMARTCARD_DISABLE

Description:

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. The Handle Instance can be UARTx where x: 1, 2, 3 to select the USART peripheral

Return value:

- None:

IS_SMARTCARD_BAUDRATE

Description:

- Check the Baud rate range.

Parameters:

- __BAUDRATE__: Baud rate set by the configuration function.

Return value:

- Test: result (TRUE or FALSE)

IS_SMARTCARD_BLOCKLENGTH

Description:

- Check the block length range.

Parameters:

- __LENGTH__: block length.

Return value:

- Test: result (TRUE or FALSE)

IS_SMARTCARD_TIMEOUT_VALUE

Description:

- Check the receiver timeout value.

Parameters:

- __TIMEOUTVALUE__: receiver timeout value.

Return value:

- Test: result (TRUE or FALSE)

IS_SMARTCARD_AUTORETRY_COUNT

Description:

- Check the SMARTCARD autoretry counter value.

Parameters:

- __COUNT__: number of retransmissions

Return value:

- Test: result (TRUE or FALSE)

SMARTCARD guard time value LSB position in GTPR register

SMARTCARD_GTPR_GT_LSB_POS

SMARTCARD interruptions flag mask

SMARTCARD_IT_MASK

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE

SMARTCARD_IT_TXE

SMARTCARD_IT_TC

SMARTCARD_IT_RXNE

SMARTCARD_IT_ERR

SMARTCARD_IT_ORE

SMARTCARD_IT_NE

SMARTCARD_IT_FE

SMARTCARD_IT_EOB

SMARTCARD_IT_RTO

SMARTCARD Interruption Clear Flags

SMARTCARD_CLEAR_PEF	Parity Error Clear Flag
SMARTCARD_CLEAR_FEF	Framing Error Clear Flag
SMARTCARD_CLEAR_NEF	Noise detected Clear Flag
SMARTCARD_CLEAR_OREF	OverRun Error Clear Flag
SMARTCARD_CLEAR_TCF	Transmission Complete Clear Flag
SMARTCARD_CLEAR_RTOF	Receiver Time Out Clear Flag
SMARTCARD_CLEAR_EOBF	End Of Block Clear Flag

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLED

SMARTCARD_LASTBIT_ENABLED

IS_SMARTCARD_LASTBIT

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX

SMARTCARD_MODE_TX

SMARTCARD_MODE_TX_RX

IS_SMARTCARD_MODE

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE

SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE

IS_SMARTCARD_ADVFEATURE_MSBFIRST

SMARTCARD NACK Enable

SMARTCARD_NACK_ENABLED

SMARTCARD_NACK_DISABLED

IS_SMARTCARD_NACK

SMARTCARD One Bit Sampling Method

SMARTCARD_ONEBIT_SAMPLING_DISABLED

SMARTCARD_ONEBIT_SAMPLING_ENABLED

IS_SMARTCARD_ONEBIT_SAMPLING

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE

SMARTCARD_ADVFEATURE_OVERRUN_DISABLE

IS_SMARTCARD_OVERRUN

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD_PARITY_ODD

IS_SMARTCARD_PARITY

SMARTCARD Private Define

TEACK_REACK_TIMEOUT

SMARTCARD_TXDMA_TIMEOUTVALUE

SMARTCARD_TIMEOUT_VALUE

USART_CR1_FIELDS

USART_CR2_CLK_FIELDS

USART_CR2_FIELDS

USART_CR3_FIELDS

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST	Receive Data flush Request
SMARTCARD_TXDATA_FLUSH_REQUEST	Transmit data flush Request

IS_SMARTCARD_REQUEST_PARAMETER

SMARTCARD block length LSB position in RTOR register

SMARTCARD_RTOR_BLEN_LSB_POS

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE

SMARTCARD_ADVFEATURE_RXINV_ENABLE

IS_SMARTCARD_ADVFEATURE_RXINV

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE

SMARTCARD_ADVFEATURE_SWAP_ENABLE

IS_SMARTCARD_ADVFEATURE_SWAP

SMARTCARD Stop Bits

SMARTCARD_STOPBITS_1_5

IS_SMARTCARD_STOPBITS

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLED

SMARTCARD_TIMEOUT_ENABLED

IS_SMARTCARD_TIMEOUT

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE

SMARTCARD_ADVFEATURE_TXINV_ENABLE

IS_SMARTCARD_ADVFEATURE_TXINV

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

IS_SMARTCARD_WORD_LENGTH

44 HAL SMARTCARD Extension Driver

44.1 SMARTCARDEx Firmware driver API description

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

44.1.1 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARDEx_BlockLength_Config() API allows to configure the Block Length on the fly
- HAL_SMARTCARDEx_TimeOut_Config() API allows to configure the receiver timeout value on the fly
- HAL_SMARTCARDEx_EnableReceiverTimeOut() API enables the receiver timeout feature
- HAL_SMARTCARDEx_DisableReceiverTimeOut() API disables the receiver timeout feature
- ***HAL_SMARTCARDEx_BlockLength_Config()***
- ***HAL_SMARTCARDEx_TimeOut_Config()***
- ***HAL_SMARTCARDEx_EnableReceiverTimeOut()***
- ***HAL_SMARTCARDEx_DisableReceiverTimeOut()***

44.1.2 HAL_SMARTCARDEx_BlockLength_Config

Function Name	**void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)**
Function Description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	• **hsmartcard:** SMARTCARD handle • **BlockLength:** SMARTCARD block length (8-bit long at most)
Return values	• None

44.1.3 HAL_SMARTCARDEx_TimeOut_Config

Function Name	**void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)**
Function Description	Update on the fly the receiver timeout value in RTOR register.
Parameters	• **hsmartcard:** SMARTCARD handle • **TimeOutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFFF.
Return values	• None

44.1.4 HAL_SMARTCARDEx_EnableReceiverTimeOut

Function Name	**HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	Enable the SMARTCARD receiver timeout feature.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• HAL status

44.1.5 HAL_SMARTCARDEx_DisableReceiverTimeOut

Function Name	**HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)**
Function Description	Disable the SMARTCARD receiver timeout feature.
Parameters	• **hsmartcard:** SMARTCARD handle
Return values	• HAL status

44.2 SMARTCARDEx Firmware driver defines

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

44.2.1 SMARTCARDEx

SMARTCARDEx

SMARTCARD Extended Exported Macros

__HAL_SMARTCARD_GETCLOCKSOURCE

Description:

- Reports the SMARTCARD clock source.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle
- __CLOCKSOURCE__: output variable

Return value:

- the: SMARTCARD clocking source, written in __CLOCKSOURCE__.

45 HAL SMBUS Generic Driver

45.1 SMBUS Firmware driver registers structures

45.1.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the stm32f3xx_hal_smbus.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***
- ***uint32_t PacketErrorCheckMode***
- ***uint32_t PeripheralMode***
- ***uint32_t SMBusTimeout***

Field Documentation

- ***uint32_t SMBUS_InitTypeDef::Timing***
  Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32_t SMBUS_InitTypeDef::AnalogFilter***
  Specifies if Analog Filter is enable or not. This parameter can be a a value of ***SMBUS_Analog_Filter***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t SMBUS_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode for master is selected. This parameter can be a value of ***SMBUS_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***SMBUS_dual_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of ***SMBUS_own_address2_masks***.
- ***uint32_t SMBUS_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***SMBUS_general_call_addressing_mode***.

- ***uint32_t SMBUS_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***SMBUS_nostretch_mode***
- ***uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode***
  Specifies if Packet Error Check mode is selected. This parameter can be a value of ***SMBUS_packet_error_check_mode***
- ***uint32_t SMBUS_InitTypeDef::PeripheralMode***
  Specifies which mode of Periphal is selected. This parameter can be a value of ***SMBUS_peripheral_mode***
- ***uint32_t SMBUS_InitTypeDef::SMBusTimeout***
  Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual.

45.1.2 SMBUS_HandleTypeDef

SMBUS_HandleTypeDef is defined in the stm32f3xx_hal_smbus.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***SMBUS_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO HAL_SMBUS_StateTypeDef PreviousState***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMBUS_StateTypeDef State***
- ***__IO HAL_SMBUS_ErrorTypeDef ErrorCode***

Field Documentation

- ***I2C_TypeDef* SMBUS_HandleTypeDef::Instance***
  SMBUS registers base address .
- ***SMBUS_InitTypeDef SMBUS_HandleTypeDef::Init***
  SMBUS communication parameters.
- ***uint8_t* SMBUS_HandleTypeDef::pBuffPtr***
  Pointer to SMBUS transfer buffer.
- ***uint16_t SMBUS_HandleTypeDef::XferSize***
  SMBUS transfer size.
- ***__IO uint16_t SMBUS_HandleTypeDef::XferCount***
  SMBUS transfer counter.
- ***__IO uint32_t SMBUS_HandleTypeDef::XferOptions***
  SMBUS transfer options.
- ***__IO HAL_SMBUS_StateTypeDef SMBUS_HandleTypeDef::PreviousState***
  SMBUS communication Previous state.
- ***HAL_LockTypeDef SMBUS_HandleTypeDef::Lock***
  SMBUS locking object.
- ***__IO HAL_SMBUS_StateTypeDef SMBUS_HandleTypeDef::State***
  SMBUS communication state.

- ***__IO HAL_SMBUS_ErrorTypeDef SMBUS_HandleTypeDef::ErrorCode***
  SMBUS Error code.

45.2 SMBUS Firmware driver API description

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

45.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example: SMBUS_HandleTypeDef hsmbus;
2. Initialize the SMBUS low level resources by implement the HAL_SMBUS_MspInit() API:
   a. Enable the SMBUSx interface clock
   b. SMBUS pins configuration
      – Enable the clock for the SMBUS GPIOs
      – Configure SMBUS pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      – Configure the SMBUSx interrupt priority
      – Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Adressing Mode, Dual Addressing Mode, Own Address2, Own Address2 Mask, General Call, Nostretch Mode, Peripheral Mode and Packet Error Check Mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the HAL_SMBUS_Init() API:
   – These API's configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customed HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver :

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in no-blocking mode using HAL_SMBUS_Master_Transmit_IT().
  – At transmission end of transfer, HAL_SMBUS_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMBUS_MasterTxCpltCallback().
- Receive in master/host SMBUS mode an amount of data in no-blocking mode using HAL_SMBUS_Master_Receive_IT().
  – At reception end of transfer, HAL_SMBUS_MasterRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_MasterRxCpltCallback().
- Abort a master/host SMBUS process commnunication with Interrupt using HAL_SMBUS_Master_Abort_IT().
  – The associated previous transfer callback is called at the end of abort process.
  – mean HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit.

  – mean HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive.
- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using HAL_SMBUS_Slave_Listen_IT() HAL_SMBUS_DisableListen_IT().
  – When address slave/device SMBUS match, HAL_SMBUS_SlaveAddrCallback() is executed and user can add his own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).
  – At Listen mode end, HAL_SMBUS_SlaveListenCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveListenCpltCallback().
- Transmit in slave/device SMBUS mode an amount of data in no-blocking mode using HAL_SMBUS_Slave_Transmit_IT()
  – At transmission end of transfer, HAL_SMBUS_SlaveTxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveTxCpltCallback().
- Receive in slave/device SMBUS mode an amount of data in no-blocking mode using HAL_SMBUS_Slave_Receive_IT()
  – At reception end of transfer, HAL_SMBUS_SlaveRxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback().
- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT() HAL_SMBUS_DisableAlert_IT().
  – When SMBUS Alert is generated, HAL_SMBUS_ErrorCallback() is executed and user can add his own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function HAL_SMBUS_GetError().
- Get HAL state machine or error values using HAL_SMBUS_GetState() or HAL_SMBUS_GetError().
- In case of transfer Error, HAL_SMBUS_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMBUS_ErrorCallback to check the Error Code using function HAL_SMBUS_GetError().

SMBUS HAL driver macros list

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

- __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- __HAL_SMBUS_GET_FLAG: Check whether the specified SMBUS flag is set or not
- __HAL_SMBUS_CLEAR_FLAG: Clear the specified SMBUS pending flag
- __HAL_SMBUS_ENABLE_IT: Enable the specified SMBUS interrupt
- __HAL_SMBUS_DISABLE_IT: Disable the specified SMBUS interrupt

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

45.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialiaze the SMBUSx peripheral:

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filer mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode
- Call the function HAL_SMBUS_DeInit() to restore the default configuration of the selected SMBUSx peripheral.
- ***HAL_SMBUS_Init()***
- ***HAL_SMBUS_DeInit()***
- ***HAL_SMBUS_MspInit()***
- ***HAL_SMBUS_MspDeInit()***

45.2.3 IO operation functions

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

1. Blocking mode function to check if device is ready for usage is :
   - HAL_SMBUS_IsDeviceReady()
2. There is only one mode of transfer:
   - No-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. No-Blocking mode functions with Interrupt are :
   - HAL_SMBUS_Master_Transmit_IT()
   - HAL_SMBUS_Master_Receive_IT()
   - HAL_SMBUS_Slave_Transmit_IT()
   - HAL_SMBUS_Slave_Receive_IT()
   - HAL_SMBUS_Slave_Listen_IT() or alias HAL_SMBUS_EnableListen_IT()
   - HAL_SMBUS_DisableListen_IT()
   - HAL_SMBUS_EnableAlert_IT()
   - HAL_SMBUS_DisableAlert_IT()
4. A set of Transfer Complete Callbacks are provided in No-Blocking mode:
   - HAL_SMBUS_MasterTxCpltCallback()
   - HAL_SMBUS_MasterRxCpltCallback()
   - HAL_SMBUS_SlaveTxCpltCallback()
   - HAL_SMBUS_SlaveRxCpltCallback()
   - HAL_SMBUS_SlaveAddrCallback() or alias HAL_SMBUS_AddrCallback()
   - HAL_SMBUS_SlaveListenCpltCallback() or alias HAL_SMBUS_ListenCpltCallback()
   - HAL_SMBUS_ErrorCallback()

- ***HAL_SMBUS_Master_Transmit_IT()***
- ***HAL_SMBUS_Master_Receive_IT()***

- ***HAL_SMBUS_Master_Abort_IT()***
- ***HAL_SMBUS_Slave_Transmit_IT()***
- ***HAL_SMBUS_Slave_Receive_IT()***
- ***HAL_SMBUS_Slave_Listen_IT()***
- ***HAL_SMBUS_DisableListen_IT()***
- ***HAL_SMBUS_EnableAlert_IT()***
- ***HAL_SMBUS_DisableAlert_IT()***
- ***HAL_SMBUS_IsDeviceReady()***
- ***HAL_SMBUS_EV_IRQHandler()***
- ***HAL_SMBUS_ER_IRQHandler()***
- ***HAL_SMBUS_MasterTxCpltCallback()***
- ***HAL_SMBUS_MasterRxCpltCallback()***
- ***HAL_SMBUS_SlaveTxCpltCallback()***
- ***HAL_SMBUS_SlaveRxCpltCallback()***
- ***HAL_SMBUS_SlaveAddrCallback()***
- ***HAL_SMBUS_SlaveListenCpltCallback()***
- ***HAL_SMBUS_ErrorCallback()***

45.2.4 Peripheral State and Errors functions

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

- ***HAL_SMBUS_GetState()***
- ***HAL_SMBUS_GetError()***

45.2.5 HAL_SMBUS_Init

Function Name	**HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)**
Function Description	Initialize the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and create the associated handle.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• HAL status

45.2.6 HAL_SMBUS_DeInit

Function Name	**HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)**
Function Description	DeInitialize the SMBUS peripheral.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• HAL status

45.2.7 HAL_SMBUS_MspInit

Function Name	**void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)**
Function Description	SMBUS MSP Init.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.8 HAL_SMBUS_MspDeInit

Function Name	**void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)**
Function Description	SMBUS MSP DeInit.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.9 HAL_SMBUS_Master_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function Description	Transmit in master/host SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DevAddress:** Target device address • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• HAL status

45.2.10 HAL_SMBUS_Master_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function Description	Receive in master/host SMBUS mode an amount of data in no-

	blocking mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DevAddress:** Target device address • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• HAL status

45.2.11 HAL_SMBUS_Master_Abort_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)**
Function Description	Abort a master/host SMBUS process commnunication with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DevAddress:** Target device address
Return values	• HAL status
Notes	• This abort can be called only if state is ready

45.2.12 HAL_SMBUS_Slave_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**
Function Description	Transmit in slave/device SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• HAL status

45.2.13 HAL_SMBUS_Slave_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)**

Function Description	Receive in slave/device SMBUS mode an amount of data in no-blocking mode with Interrupt.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **pData:** Pointer to data buffer • **Size:** Amount of data to be sent • **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition
Return values	• HAL status

45.2.14 HAL_SMBUS_Slave_Listen_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_Slave_Listen_IT (SMBUS_HandleTypeDef * hsmbus)**
Function Description	This function enable the Address listen mode.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• HAL status

45.2.15 HAL_SMBUS_DisableListen_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)**
Function Description	This function disable the Address listen mode.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• HAL status

45.2.16 HAL_SMBUS_EnableAlert_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)**
Function Description	This function enable the SMBUS alert mode.
Parameters	• **hsmbus:** pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	• HAL status

45.2.17 HAL_SMBUS_DisableAlert_IT

Function Name	**HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)**
Function Description	This function disable the SMBUS alert mode.
Parameters	• **hsmbus:** pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
Return values	• HAL status

45.2.18 HAL_SMBUS_IsDeviceReady

Function Name	**HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)**
Function Description	Check if target device is ready for communication.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **DevAddress:** Target device address • **Trials:** Number of trials • **Timeout:** Timeout duration
Return values	• HAL status

45.2.19 HAL_SMBUS_EV_IRQHandler

Function Name	**void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)**
Function Description	This function handles SMBUS event interrupt request.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.20 HAL_SMBUS_ER_IRQHandler

Function Name	**void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)**
Function Description	This function handles SMBUS error interrupt request.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.21 HAL_SMBUS_MasterTxCpltCallback

Function Name	**void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function Description	Master Tx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.22 HAL_SMBUS_MasterRxCpltCallback

Function Name	**void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function Description	Master Rx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.23 HAL_SMBUS_SlaveTxCpltCallback

Function Name	**void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function Description	Slave Tx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.24 HAL_SMBUS_SlaveRxCpltCallback

Function Name	**void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function Description	Slave Rx Transfer completed callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.25 HAL_SMBUS_SlaveAddrCallback

Function Name	**void HAL_SMBUS_SlaveAddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)**
Function Description	Slave Address Match callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS. • **TransferDirection:** Master request Transfer Direction (Write/Read) • **AddrMatchCode:** Address Match Code
Return values	• None

45.2.26 HAL_SMBUS_SlaveListenCpltCallback

Function Name	**void HAL_SMBUS_SlaveListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)**
Function Description	Listen Complete callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.27 HAL_SMBUS_ErrorCallback

Function Name	**void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)**
Function Description	SMBUS error callback.
Parameters	• **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• None

45.2.28 HAL_SMBUS_GetState

Function Name	**HAL_SMBUS_StateTypeDef HAL_SMBUS_GetState (SMBUS_HandleTypeDef * hsmbus)**
Function Description	Return the SMBUS state.
Parameters	• **hsmbus:** pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• HAL state

45.2.29 HAL_SMBUS_GetError

Function Name	**uint32_t HAL_SMBUS_GetError (SMBUS_HandleTypeDef * hsmbus)**
Function Description	Return the SMBUS error code.
Parameters	• **hsmbus:** pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
Return values	• SMBUS Error Code

45.3 SMBUS Firmware driver defines

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

45.3.1 SMBUS

SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS_ADDRESSINGMODE_10BIT

IS_SMBUS_ADDRESSING_MODE

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLED

SMBUS_ANALOGFILTER_DISABLED

IS_SMBUS_ANALOG_FILTER

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLED

SMBUS_DUALADDRESS_ENABLED

IS_SMBUS_DUAL_ADDRESS

Input and Output operation functions

HAL_SMBUS_EnableListen_IT

HAL_SMBUS_AddrCallback

HAL_SMBUS_ListenCpltCallback

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE	**Description:** • Reset SMBUS handle state. **Parameters:** • __HANDLE__: SMBUS handle. **Return value:** • None:

__HAL_SMBUS_ENABLE_IT

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI: Errors interrupt enable
  - SMBUS_IT_TCI: Transfer complete interrupt enable
  - SMBUS_IT_STOPI: STOP detection interrupt enable
  - SMBUS_IT_NACKI: NACK received interrupt enable
  - SMBUS_IT_ADDRI: Address match interrupt enable
  - SMBUS_IT_RXI: RX interrupt enable
  - SMBUS_IT_TXI: TX interrupt enable

Return value:

- None:

__HAL_SMBUS_DISABLE_IT

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SMBUS_IT_ERRI: Errors interrupt enable
  - SMBUS_IT_TCI: Transfer complete interrupt enable
  - SMBUS_IT_STOPI: STOP detection interrupt enable
  - SMBUS_IT_NACKI: NACK received interrupt enable
  - SMBUS_IT_ADDRI: Address match interrupt enable
  - SMBUS_IT_RXI: RX interrupt

enable
- SMBUS_IT_TXI: TX interrupt enable

Return value:

- None:

__HAL_SMBUS_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  - SMBUS_IT_ERRI: Errors interrupt enable
  - SMBUS_IT_TCI: Transfer complete interrupt enable
  - SMBUS_IT_STOPI: STOP detection interrupt enable
  - SMBUS_IT_NACKI: NACK received interrupt enable
  - SMBUS_IT_ADDRI: Address match interrupt enable
  - SMBUS_IT_RXI: RX interrupt enable
  - SMBUS_IT_TXI: TX interrupt enable

Return value:

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

SMBUS_FLAG_MASK

Description:

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

Parameters:

- __HANDLE__: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS_FLAG_TXE: Transmit data register empty
  - SMBUS_FLAG_TXIS: Transmit

interrupt status
  - SMBUS_FLAG_RXNE: Receive data register not empty
  - SMBUS_FLAG_ADDR: Address matched (slave mode)
  - SMBUS_FLAG_AF: NACK received flag
  - SMBUS_FLAG_STOPF: STOP detection flag
  - SMBUS_FLAG_TC: Transfer complete (master mode)
  - SMBUS_FLAG_TCR: Transfer complete reload
  - SMBUS_FLAG_BERR: Bus error
  - SMBUS_FLAG_ARLO: Arbitration lost
  - SMBUS_FLAG_OVR: Overrun/Underrun
  - SMBUS_FLAG_PECERR: PEC error in reception
  - SMBUS_FLAG_TIMEOUT: Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT: SMBus alert
  - SMBUS_FLAG_BUSY: Bus busy
  - SMBUS_FLAG_DIR: Transfer direction (slave mode)

Return value:

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

__HAL_SMBUS_GET_FLAG

__HAL_SMBUS_CLEAR_FLAG

Description:

- Clears the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle. This parameter can be SMBUS where x: 1 or 2 to select the SMBUS peripheral.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - SMBUS_FLAG_ADDR: Address matched (slave mode)
  - SMBUS_FLAG_AF: NACK received flag
  - SMBUS_FLAG_STOPF: STOP detection flag
  - SMBUS_FLAG_BERR: Bus error
  - SMBUS_FLAG_ARLO: Arbitration lost

- SMBUS_FLAG_OVR: Overrun/Underrun
- SMBUS_FLAG_PECERR: PEC error in reception
- SMBUS_FLAG_TIMEOUT: Timeout or Tlow detection flag
- SMBUS_FLAG_ALERT: SMBus alert

Return value:

- None:

__HAL_SMBUS_ENABLE

Description:

- Enable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None:

__HAL_SMBUS_DISABLE

Description:

- Disable the specified SMBUS peripheral.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None:

__HAL_SMBUS_RESET_CR1

__HAL_SMBUS_RESET_CR2

__HAL_SMBUS_GENERATE_START

__HAL_SMBUS_GET_ADDR_MATCH

__HAL_SMBUS_GET_DIR

__HAL_SMBUS_GET_STOP_MODE

__HAL_SMBUS_GET_PEC_MODE

__HAL_SMBUS_GET_ALERT_ENABLED

__HAL_SMBUS_GENERATE_NACK

IS_SMBUS_OWN_ADDRESS1

IS_SMBUS_OWN_ADDRESS2

SMBUS Flag definition

SMBUS_FLAG_TXE

SMBUS_FLAG_TXIS

SMBUS_FLAG_RXNE

SMBUS_FLAG_ADDR

SMBUS_FLAG_AF

SMBUS_FLAG_STOPF

SMBUS_FLAG_TC

SMBUS_FLAG_TCR

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLED

SMBUS_GENERALCALL_ENABLED

IS_SMBUS_GENERAL_CALL

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLED

SMBUS_NOSTRETCH_ENABLED

IS_SMBUS_NO_STRETCH

SMBUS own address2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

IS_SMBUS_OWN_ADDRESS2_MASK

SMBUS packet error check mode

SMBUS_PEC_DISABLED

SMBUS_PEC_ENABLED

IS_SMBUS_PEC

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

IS_SMBUS_PERIPHERAL_MODE

SMBUS Private Define

TIMING_CLEAR_MASK

HAL_TIMEOUT_ADDR

HAL_TIMEOUT_BUSY

HAL_TIMEOUT_DIR

HAL_TIMEOUT_RXNE

HAL_TIMEOUT_STOPF

HAL_TIMEOUT_TC

HAL_TIMEOUT_TCR

HAL_TIMEOUT_TXIS

MAX_NBYTE_SIZE

SMBUS Private Macros

__SMBUS_GET_ISR_REG

__SMBUS_CHECK_FLAG

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

IS_SMBUS_TRANSFER_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

IS_SMBUS_TRANSFER_REQUEST

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

IS_SMBUS_TRANSFER_OPTIONS_REQUEST

46 HAL SPI Generic Driver

46.1 SPI Firmware driver registers structures

46.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32f3xx_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***
- ***uint32_t CRCLength***
- ***uint32_t NSSPMode***

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
  Specifies the SPI operating mode. This parameter can be a value of ***SPI_mode***
- ***uint32_t SPI_InitTypeDef::Direction***
  Specifies the SPI bidirectional mode state. This parameter can be a value of ***SPI_Direction***
- ***uint32_t SPI_InitTypeDef::DataSize***
  Specifies the SPI data size. This parameter can be a value of ***SPI_data_size***
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint32_t SPI_InitTypeDef::CLKPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_management***
- ***uint32_t SPI_InitTypeDef::BaudRatePrescaler***
  Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler***
  Note: The communication clock is derived from the master clock. The slave clock does not need to be set.
- ***uint32_t SPI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_transmission***

- ***uint32_t SPI_InitTypeDef::TIMode***
  Specifies if the TI mode is enabled or not . This parameter can be a value of ***SPI_TI_mode***
- ***uint32_t SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_CRC_Calculation***
- ***uint32_t SPI_InitTypeDef::CRCPolynomial***
  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0 and Max_Data = 65535
- ***uint32_t SPI_InitTypeDef::CRCLength***
  Specifies the CRC Length used for the CRC calculation. CRC Length is only used with Data8 and Data16, not other data size This parameter must 0 or 1 or 2
- ***uint32_t SPI_InitTypeDef::NSSPMode***
  Specifies whether the NSSP signal is enabled or not . This parameter can be a value of ***SPI_NSSP_Mode*** This mode is activated by the NSSP bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx_CR1 CPHA = 0, CPOL setting is ignored)..

46.1.2 __SPI_HandleTypeDef

__SPI_HandleTypeDef is defined in the stm32f3xx_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***
- ***uint32_t CRCSize***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***HAL_SPI_StateTypeDef State***
- ***HAL_SPI_ErrorTypeDef ErrorCode***

Field Documentation

- ***SPI_TypeDef* __SPI_HandleTypeDef::Instance***
- ***SPI_InitTypeDef __SPI_HandleTypeDef::Init***
- ***uint8_t* __SPI_HandleTypeDef::pTxBuffPtr***
- ***uint16_t __SPI_HandleTypeDef::TxXferSize***
- ***uint16_t __SPI_HandleTypeDef::TxXferCount***
- ***uint8_t* __SPI_HandleTypeDef::pRxBuffPtr***
- ***uint16_t __SPI_HandleTypeDef::RxXferSize***
- ***uint16_t __SPI_HandleTypeDef::RxXferCount***

- ***uint32_t __SPI_HandleTypeDef::CRCSize***
- ***void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
- ***void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
- ***HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
- ***HAL_SPI_ErrorTypeDef __SPI_HandleTypeDef::ErrorCode***

46.2 SPI Firmware driver API description

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

46.2.1 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 implement 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 interface clock using
      – Configure the DMA handle parameters
      – Configure the DMA Tx or Rx channel
      – Associate the initilalized hdma_tx handle to the hspi DMA Tx or Rx handle
      – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
   – These APIs configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customed HAL_SPI_MspInit(&hspi) API.

Using the HAL it is not possible to reach all supported SPI frequency with the differents SPI Modes, the following table resume the max SPI frequency reached with data size 8bits/16bits:

Table 28: Maximum SPI frequency vs data size

Process	Transfer mode	2 lines, fullduplex		2 line, Rx only		1 line	
		Master	Slave	Master	Slave	Master	Slave
Tx/Rx	Polling	$f_{CPU}/4$	$f_{CPU}/8$	NA	NA	NA	NA

Process	Transfer mode	2 lines, fullduplex		2 line, Rx only		1 line	
		Master	Slave	Master	Slave	Master	Slave
	Interrupt	$f_{CPU}/4$	$f_{CPU}/16$	NA	NA	NA	NA
	DMA	$f_{CPU}/2$	$f_{CPU}/2$	NA	NA	NA	NA
Rx	Polling	$f_{CPU}/4$	$f_{CPU}/8$	$f_{CPU}/16$	$f_{CPU}/8$	$f_{CPU}/8$	$f_{CPU}/8$
	Interrupt	$f_{CPU}/8$	$f_{CPU}/16$	$f_{CPU}/8$	$f_{CPU}/8$	$f_{CPU}/8$	$f_{CPU}/4$
	DMA	$f_{CPU}/4$	$f_{CPU}/2$	$f_{CPU}/2$	$f_{CPU}/16$	$f_{CPU}/2$	$f_{CPU}/16$
Tx	Polling	$f_{CPU}/8$	$f_{CPU}/2$	NA	NA	$f_{CPU}/8$	$f_{CPU}/8$
	Interrupt	$f_{CPU}/2$	$f_{CPU}/4$	NA	NA	$f_{CPU}/16$	$f_{CPU}/8$
	DMA	$f_{CPU}/2$	$f_{CPU}/2$	NA	NA	$f_{CPU}/8$	$f_{CPU}/16$

The max SPI frequency depend on SPI data size (4bits, 5bits,..., 8bits,...15bits, 16bits), SPI mode(2 Lines fullduplex, 2 lines RxOnly, 1 line TX/RX) and Process mode (Polling, IT, DMA).

- TX/RX processes are HAL_SPI_TransmitReceive(), HAL_SPI_TransmitReceive_IT() and HAL_SPI_TransmitReceive_DMA()
- RX processes are HAL_SPI_Receive(), HAL_SPI_Receive_IT() and HAL_SPI_Receive_DMA()
- TX processes are HAL_SPI_Transmit(), HAL_SPI_Transmit_IT() and HAL_SPI_Transmit_DMA()

46.2.2 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
  - CRC Length, used only with Data8 and Data16
  - FIFO reception threshold
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx periperal.

- ***HAL_SPI_Init()***
- ***HAL_SPI_DeInit()***
- ***HAL_SPI_MspInit()***
- ***HAL_SPI_MspDeInit()***
- ***HAL_SPI_InitExtended()***

46.2.3 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.
   - 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 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. Blocking mode API's are :
   - HAL_SPI_Transmit()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
   - HAL_SPI_Transmit_IT()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive_IT() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceive_IT()in full duplex mode
   - HAL_SPI_IRQHandler()
4. Non-Blocking mode functions with DMA are :
   - HAL_SPI_Transmit_DMA()in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_Receive_DMA() in 1Line (simplex) and 2Lines (full duplex) mode
   - HAL_SPI_TransmitReceie_DMA() in full duplex mode
5. A set of Transfer Complete Callbacks are provided in No_Blocking mode:
   - HAL_SPI_TxCpltCallback()
   - HAL_SPI_RxCpltCallback()
   - HAL_SPI_ErrorCallback()
   - HAL_SPI_TxRxCpltCallback()

- ***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_IRQHandler()***
- ***HAL_SPI_TxCpltCallback()***
- ***HAL_SPI_RxCpltCallback()***
- ***HAL_SPI_TxRxCpltCallback()***

- ***HAL_SPI_ErrorCallback()***

46.2.4 Peripheral Control 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_GetState()***

46.2.5 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

46.2.6 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

46.2.7 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.

46.2.8 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.

46.2.9 HAL_SPI_InitExtended

Function Name	**HAL_StatusTypeDef HAL_SPI_InitExtended (SPI_HandleTypeDef * hspi)**
Function Description	

46.2.10 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

46.2.11 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

46.2.12 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

46.2.13 HAL_SPI_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function Description	Transmit an amount of data in Non-Blocking mode with Interrupt.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

46.2.14 HAL_SPI_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function Description	Receive an amount of data in Non-Blocking mode with Interrupt.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

46.2.15 HAL_SPI_TransmitReceive_IT

Function Name	**HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**
Function Description	Transmit and Receive an amount of data in Non-Blocking mode with Interrupt.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pTxData:** pointer to transmission data buffer • **pRxData:** pointer to reception data buffer to be • **Size:** amount of data to be sent
Return values	• HAL status

46.2.16 HAL_SPI_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_SPI_Transmit_DMA**

	(SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in Non-Blocking mode with DMA.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

46.2.17 HAL_SPI_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**
Function Description	Receive an amount of data in Non-Blocking mode with DMA.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

46.2.18 HAL_SPI_TransmitReceive_DMA

Function Name	**HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)**
Function Description	Transmit and Receive an amount of data in Non-Blocking mode with DMA.
Parameters	• **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • **pTxData:** pointer to transmission data buffer • **pRxData:** pointer to reception data buffer • **Size:** amount of data to be sent
Return values	• HAL status

46.2.19 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 the specified SPI module.
Return values	• None.

46.2.20 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

46.2.21 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

46.2.22 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

46.2.23 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

46.2.24 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 • HAL state

46.3 SPI Firmware driver defines

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

46.3.1 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

IS_SPI_BAUDRATE_PRESCALER

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

IS_SPI_CPHA

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

IS_SPI_CPOL

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLED

SPI_CRCCALCULATION_ENABLED

IS_SPI_CRC_CALCULATION

SPI CRC length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_16BIT

IS_SPI_CRC_LENGTH

SPI Data size

SPI_DATASIZE_4BIT

SPI_DATASIZE_5BIT

SPI_DATASIZE_6BIT

SPI_DATASIZE_7BIT

SPI_DATASIZE_8BIT

SPI_DATASIZE_9BIT

SPI_DATASIZE_10BIT

SPI_DATASIZE_11BIT

SPI_DATASIZE_12BIT

SPI_DATASIZE_13BIT

SPI_DATASIZE_14BIT

SPI_DATASIZE_15BIT

SPI_DATASIZE_16BIT

IS_SPI_DATASIZE

SPI Direction

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

IS_SPI_DIRECTION

IS_SPI_DIRECTION_2LINES

IS_SPI_DIRECTION_2LINES_OR_1LINE

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: SPI handle.

Return value:

- None:

__HAL_SPI_ENABLE_IT

Description:

- Enables or disables 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 or disable. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable

- SPI_IT_ERR: Error interrupt enable

Return value:

- None:

__HAL_SPI_DISABLE_IT

__HAL_SPI_GET_IT_SOURCE

Description:

- Checks 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:

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

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXNE: Receive buffer not empty flag
  - SPI_FLAG_TXE: Transmit buffer empty flag
  - SPI_FLAG_CRCERR: CRC error flag
  - SPI_FLAG_MODF: Mode fault flag
  - SPI_FLAG_OVR: Overrun flag
  - SPI_FLAG_BSY: Busy flag
  - SPI_FLAG_FRE: Frame format error flag
  - SPI_FLAG_FTLVL: SPI fifo transmission level
  - SPI_FLAG_FRLVL: SPI fifo reception level

Return value:

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

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clears 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:

- Clears 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:

- Clears the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clears the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None:

__HAL_SPI_ENABLE

Description:

- 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:

__HAL_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:

__HAL_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:

__HAL_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:

IS_SPI_CRC_POLYNOMIAL

SPI FIFO reception threshold

SPI_RXFIFO_THRESHOLD

SPI_RXFIFO_THRESHOLD_QF

SPI_RXFIFO_THRESHOLD_HF

SPI Flag definition

SPI_FLAG_RXNE

SPI_FLAG_TXE

SPI_FLAG_BSY

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_OVR

SPI_FLAG_FRE

SPI_FLAG_FTLVL

SPI_FLAG_FRLVL

IS_SPI_FLAG

SPI Interrupt configuration definition

SPI_IT_TXE

SPI_IT_RXNE

SPI_IT_ERR

IS_SPI_IT

SPI mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

IS_SPI_MODE

SPI MSB LSB transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

IS_SPI_FIRST_BIT

SPI NSS pulse management

SPI_NSS_PULSE_ENABLED

SPI_NSS_PULSE_DISABLED

IS_SPI_NSSP

SPI Private Define

SPI_DEFAULT_TIMEOUT

SPI reception fifo status level

SPI_FRLVL_EMPTY

SPI_FRLVL_QUARTER_FULL

SPI_FRLVL_HALF_FULL

SPI_FRLVL_FULL

SPI Slave Select management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

IS_SPI_NSS

SPI TI mode

SPI_TIMODE_DISABLED

SPI_TIMODE_ENABLED

IS_SPI_TIMODE

SPI transmission fifo status level

SPI_FTLVL_EMPTY

SPI_FTLVL_QUARTER_FULL

SPI_FTLVL_HALF_FULL

SPI_FTLVL_FULL

47 HAL SRAM Generic Driver

47.1 SRAM Firmware driver registers structures

47.1.1 SRAM_HandleTypeDef

SRAM_HandleTypeDef is defined in the stm32f3xx_hal_sram.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdma***

Field Documentation

- ***FMC_NORSRAM_TypeDef* SRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* SRAM_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef SRAM_HandleTypeDef::Init***
  SRAM device control configuration parameters
- ***HAL_LockTypeDef SRAM_HandleTypeDef::Lock***
  SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef SRAM_HandleTypeDef::State***
  SRAM device access state
- ***DMA_HandleTypeDef* SRAM_HandleTypeDef::hdma***
  Pointer DMA handler

47.2 SRAM Firmware driver API description

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

47.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC/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 FMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC_NORSRAM_TimingTypeDef Timing and FMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()
   b. Control register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Init()
   c. Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Timing_Init()
   d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Extended_Timing_Init()
   e. Enable the SRAM device using the macro __FMC_NORSRAM_ENABLE()
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   - HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
   - HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable()/ HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

47.2.2 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()***

47.2.3 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()***

47.2.4 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()***

47.2.5 SRAM State functions

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

- ***HAL_SRAM_GetState()***

47.2.6 HAL_SRAM_Init

Function Name	**HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)**
Function Description	Performs the SRAM device initialization sequence.
Parameters	• **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • **Timing:** Pointer to SRAM control timing structure • **ExtTiming:** Pointer to SRAM extended mode timing structure
Return values	• HAL status

47.2.7 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

47.2.8 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

47.2.9 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

47.2.10 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

47.2.11 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

47.2.12 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

47.2.13 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

47.2.14 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

47.2.15 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

47.2.16 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

	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

47.2.17 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

47.2.18 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

47.2.19 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

47.2.20 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

47.2.21 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

47.2.22 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

47.3 SRAM Firmware driver defines

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

47.3.1 SRAM

SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE	**Description:** • Reset SRAM handle state. **Parameters:** • __HANDLE__: SRAM handle **Return value:** • None:

SRAM Parity Lock

__HAL_SYSCFG_BREAK_SRAMPARITY_LOCK

48 HAL TIM Generic Driver

48.1 TIM Firmware driver registers structures

48.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32f3xx_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 modethe number of half PWM period in center-aligned mode This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  Note: This parameter is valid only for TIM1 and TIM8.

48.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32f3xx_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 ***TIMEx_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.

48.1.3 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the stm32f3xx_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 ***TIMEx_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

48.1.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32f3xx_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

48.1.5 TIM_Encoder_InitTypeDef

TIM_Encoder_InitTypeDef is defined in the stm32f3xx_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

48.1.6 TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32f3xx_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 value of ***TIM_Clock_Filter***

48.1.7 TIM_ClearInputConfigTypeDef

TIM_ClearInputConfigTypeDef is defined in the stm32f3xx_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 ***TIMEx_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 value of ***TIM_ClearInput_Filter***

48.1.8 TIM_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32f3xx_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 ***TIMEx_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 value of ***TIM_Trigger_Filter***

48.1.9 TIM_HandleTypeDef

TIM_HandleTypeDef is defined in the stm32f3xx_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

48.2 TIM Firmware driver API description

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

48.2.1 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

48.2.2 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 __TIMx_CLK_ENABLE();
   b. TIM pins configuration
      - Enable the clock for the TIM GPIOs using the following function: __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()

48.2.3 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()***

48.2.4 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()*

48.2.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM OPWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.
- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.
- *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()*

48.2.6 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()*

48.2.7 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()***

48.2.8 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()***

48.2.9 IRQ handler management

This section provides Timer IRQ handler function.

- ***HAL_TIM_IRQHandler()***

48.2.10 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_SlaveConfigSynchronization_DMA()***
- ***HAL_TIM_ReadCapturedValue()***

48.2.11 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()***

48.2.12 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()***

48.2.13 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

48.2.14 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

48.2.15 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

48.2.16 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

48.2.17 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

48.2.18 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

48.2.19 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

48.2.20 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

48.2.21 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

48.2.22 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

48.2.23 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

48.2.24 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

48.2.25 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

48.2.26 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

48.2.27 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

48.2.28 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

48.2.29 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

48.2.30 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

48.2.31 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

48.2.32 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

48.2.33 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

48.2.34 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

48.2.35 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

48.2.36 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

48.2.37 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

48.2.38 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

48.2.39 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

48.2.40 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

48.2.41 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

48.2.42 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

48.2.43 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

48.2.44 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

48.2.45 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

48.2.46 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

48.2.47 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

48.2.48 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

48.2.49 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

48.2.50 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

48.2.51 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

48.2.52 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

48.2.53 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

48.2.54 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

48.2.55 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

48.2.56 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

48.2.57 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

48.2.58 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

48.2.59 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

48.2.60 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

48.2.61 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

48.2.62 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

48.2.63 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

48.2.64 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

48.2.65 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
Return values	• HAL status

48.2.66 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
Return values	• HAL status

48.2.67 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
Return values	• HAL status

48.2.68 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
Return values	• HAL status

48.2.69 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 • **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

48.2.70 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
Return values	• HAL status

48.2.71 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

48.2.72 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

48.2.73 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

48.2.74 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

48.2.75 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

48.2.76 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

48.2.77 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

48.2.78 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

48.2.79 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

48.2.80 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 TIM_EventSource_Break2: Timer Break2 event source
Return values	• None
Notes	• TIM_EventSource_Break2 isn't relevant for STM32F37xx and STM32F38xx devices

48.2.81 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

48.2.82 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

48.2.83 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

48.2.84 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

48.2.85 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

48.2.86 HAL_TIM_SlaveConfigSynchronization_DMA

Function Name	**HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_DMA (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)**
Function Description	Configures the TIM in Slave mode in DMA 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

48.2.87 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

48.2.88 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

48.2.89 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

48.2.90 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

48.2.91 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

48.2.92 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

48.2.93 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

48.2.94 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

48.2.95 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

48.2.96 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

48.2.97 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

48.2.98 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

48.2.99 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

48.3 TIM Firmware driver defines

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

48.3.1 TIM

TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_ENABLE

TIM_AUTOMATICOUTPUT_DISABLE

IS_TIM_AUTOMATIC_OUTPUT_STATE

TIM Break Input Enable

TIM_BREAK_ENABLE

TIM_BREAK_DISABLE

IS_TIM_BREAK_STATE

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

TIM_BREAKPOLARITY_HIGH

IS_TIM_BREAK_POLARITY

TIM Clear Input Filter

IS_TIM_CLEARINPUT_FILTER

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARITY_INVERTED	Polarity for ETRx pin
TIM_CLEARINPUTPOLARITY_NONINVERTED	Polarity for ETRx pin

IS_TIM_CLEARINPUT_POLARITY

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.

IS_TIM_CLEARINPUT_PRESCALER

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

TIM_CLOCKDIVISION_DIV2

TIM_CLOCKDIVISION_DIV4

IS_TIM_CLOCKDIVISION_DIV

TIM Clock Filter

IS_TIM_CLOCKFILTER

IS_TIM_DEADTIME

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

IS_TIM_CLOCKPOLARITY

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.

IS_TIM_CLOCKPRESCALER

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

IS_TIM_CLOCKSOURCE

TIM Counter Mode

TIM_COUNTERMODE_UP

TIM_COUNTERMODE_DOWN

TIM_COUNTERMODE_CENTERALIGNED1

TIM_COUNTERMODE_CENTERALIGNED2

TIM_COUNTERMODE_CENTERALIGNED3

IS_TIM_COUNTER_MODE

TIM Extended DMA Base Address

TIM_DMABase_CR1

TIM_DMABase_CR2

TIM_DMABase_SMCR

TIM_DMABase_DIER

TIM_DMABase_SR

TIM_DMABase_EGR

TIM_DMABase_CCMR1

TIM_DMABase_CCMR2

TIM_DMABase_CCER

TIM_DMABase_CNT

TIM_DMABase_PSC

TIM_DMABase_ARR

TIM_DMABase_RCR

TIM_DMABase_CCR1

TIM_DMABase_CCR2

TIM_DMABase_CCR3

TIM_DMABase_CCR4

TIM_DMABase_BDTR

TIM_DMABase_DCR

TIM_DMABase_CCMR3

TIM_DMABase_CCR5

TIM_DMABase_CCR6

TIM_DMABase_OR

IS_TIM_DMA_BASE

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

IS_TIM_DMA_LENGTH

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

IS_TIM_DMA_SOURCE

TIM Encoder Mode

TIM_ENCODERMODE_TI1

TIM_ENCODERMODE_TI2

TIM_ENCODERMODE_TI12

IS_TIM_ENCODER_MODE

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED	Polarity for ETR source
TIM_ETRPOLARITY_NONINVERTED	Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1	No prescaler is used
TIM_ETRPRESCALER_DIV2	ETR input source is divided by 2
TIM_ETRPRESCALER_DIV4	ETR input source is divided by 4
TIM_ETRPRESCALER_DIV8	ETR input source is divided by 8

TIM Extended Event Source

TIM_EventSource_Update	Reinitialize the counter and generates an update of the registers
TIM_EventSource_CC1	A capture/compare event is generated on channel 1
TIM_EventSource_CC2	A capture/compare event is generated on channel 2
TIM_EventSource_CC3	A capture/compare event is generated on channel 3
TIM_EventSource_CC4	A capture/compare event is generated on channel 4
TIM_EventSource_COM	A commutation event is generated
TIM_EventSource_Trigger	A trigger event is generated
TIM_EventSource_Break	A break event is generated
TIM_EventSource_Break2	A break 2 event is generated

IS_TIM_EVENT_SOURCE

TIM Exported Constants

TIM_COMMUTATION_TRGI

TIM_COMMUTATION_SOFTWARE

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:

CCER_CCxE_MASK

CCER_CCxNE_MASK

__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

__HAL_TIM_ENABLE_DMA

__HAL_TIM_DISABLE_IT

__HAL_TIM_DISABLE_DMA

__HAL_TIM_GET_FLAG

__HAL_TIM_CLEAR_FLAG

__HAL_TIM_GET_ITSTATUS

__HAL_TIM_CLEAR_IT

__HAL_TIM_DIRECTION_STATUS

__HAL_TIM_PRESCALER

__HAL_TIM_SetICPrescalerValue

__HAL_TIM_ResetICPrescalerValue

__HAL_TIM_SetCounter

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_GetCounter

Description:

- Gets the TIM Counter Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None:

__HAL_TIM_SetAutoreload

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_GetAutoreload

Description:

- Gets the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None:

__HAL_TIM_SetClockDivision

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_GetClockDivision

Description:

- Gets the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None:

__HAL_TIM_SetICPrescaler

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_GetICPrescaler

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:

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

IS_TIM_FLAG

TIM Input Capture Value

IS_TIM_IC_FILTER

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

TIM_ICPOLARITY_FALLING

TIM_ICPOLARITY_BOTHEDGE

IS_TIM_IC_POLARITY

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

IS_TIM_IC_PRESCALER

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

IS_TIM_IC_SELECTION

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

IS_TIM_IT

IS_TIM_GET_IT

TIM Lock Configuration

TIM_LOCKLEVEL_OFF

TIM_LOCKLEVEL_1

TIM_LOCKLEVEL_2

TIM_LOCKLEVEL_3

IS_TIM_LOCK_LEVEL

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

IS_TIM_TRGO_SOURCE

TIM Master/Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

TIM_MASTERSLAVEMODE_DISABLE

IS_TIM_MSM_STATE

TIM One Pulse Mode

TIM_OPMODE_SINGLE

TIM_OPMODE_REPETITIVE

IS_TIM_OPM_MODE

TIM Off-state Selection for Idle Mode

TIM_OSSI_ENABLE

TIM_OSSI_DISABLE

IS_TIM_OSSI_STATE

TIM Off-state Selection for Run Mode

TIM_OSSR_ENABLE

TIM_OSSR_DISABLE

IS_TIM_OSSR_STATE

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

TIM_OCIDLESTATE_RESET

IS_TIM_OCIDLE_STATE

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

TIM_OCNIDLESTATE_RESET

IS_TIM_OCNIDLE_STATE

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

TIM_OCNPOLARITY_LOW

IS_TIM_OCN_POLARITY

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

TIM_OUTPUTNSTATE_ENABLE

IS_TIM_OUTPUTN_STATE

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

IS_TIM_OC_POLARITY

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

TIM_OUTPUTSTATE_ENABLE

IS_TIM_OUTPUT_STATE

TIM Output Fast State

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

IS_TIM_FAST_STATE

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

TIM_TI1SELECTION_XORCOMBINATION

IS_TIM_TI1SELECTION

TIM Trigger Filter

IS_TIM_TRIGGERFILTER

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
IS_TIM_TRIGGERPOLARITY	

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.
IS_TIM_TRIGGERPRESCALER	

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

IS_TIM_TRIGGER_SELECTION

IS_TIM_INTERNAL_TRIGGER_SELECTION

IS_TIM_INTERNAL_TRIGGEREVENT_SELECTION

49 HAL TIM Extension Driver

49.1 TIMEx Firmware driver registers structures

49.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the stm32f3xx_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

49.1.2 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32f3xx_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 BreakFilter***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  TIM off state in run mode This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode***
  TIM off state in IDLE mode This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  TIM Lock level This parameter can be a value of ***TIM_Lock_level***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime***
  TIM dead Time This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState***
  TIM Break State This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  TIM Break input polarity This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakFilter***
  Specifies the brek input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2State***
  TIM Break2 State This parameter can be a value of ***TIMEx_Break2_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Polarity***
  TIM Break2 input polarity This parameter can be a value of ***TIMEx_Break2_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Filter***
  TIM break2 input filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  TIM Automatic Output Enable state This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

49.1.3 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32f3xx_hal_tim_ex.h

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterOutputTrigger2***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***
  Trigger output (TRGO) selection This parameter can be a value of ***TIM_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger2***
  Trigger output2 (TRGO2) selection This parameter can be a value of ***TIMEx_Master_Mode_Selection_2***

- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***

49.2 TIMEx Firmware driver API description

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

49.2.1 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

49.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
   - Complementary Output Compare : HAL_TIM_OC_MspInit()
   - Complementary PWM generation : HAL_TIM_PWM_MspInit()
   - Complementary One-pulse mode output : HAL_TIM_OnePulse_MspInit()
   - Hall Sensor output : HAL_TIM_HallSensor_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __TIMx_CLK_ENABLE();
   b. TIM pins configuration
      - Enable the clock for the TIM GPIOs using the following function: __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().

49.2.3 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()***

49.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare.
- Stop the Complementary Output Compare.
- Start the Complementary Output Compare and enable interrupts.
- Stop the Complementary Output Compare and disable interrupts.
- Start the Complementary Output Compare and enable DMA transfers.
- Stop the Complementary Output Compare 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()***

49.2.5 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()***

49.2.6 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()***

49.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Output channels for OC and PWM mode.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.
- Enable or disable channel grouping
- ***HAL_TIMEx_ConfigCommutationEvent()***
- ***HAL_TIMEx_ConfigCommutationEvent_IT()***
- ***HAL_TIMEx_ConfigCommutationEvent_DMA()***
- ***HAL_TIM_OC_ConfigChannel()***
- ***HAL_TIM_PWM_ConfigChannel()***
- ***HAL_TIM_ConfigOCrefClear()***
- ***HAL_TIMEx_MasterConfigSynchronization()***

- *HAL_TIMEx_ConfigBreakDeadTime()*
- *HAL_TIMEx_RemapConfig()*
- *HAL_TIMEx_GroupChannel5()*

49.2.8 Extended Callbacks functions

This section provides Extended TIM callback functions:

- Timer Commutation callback
- Timer Break callback
- *HAL_TIMEx_CommutationCallback()*
- *HAL_TIMEx_BreakCallback()*

49.2.9 Extended Peripheral State functions

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

- *HAL_TIMEx_HallSensor_GetState()*

49.2.10 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

49.2.11 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

49.2.12 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

49.2.13 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

49.2.14 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

49.2.15 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

49.2.16 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

49.2.17 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

49.2.18 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

49.2.19 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

49.2.20 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

49.2.21 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

49.2.22 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

49.2.23 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

49.2.24 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

49.2.25 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

49.2.26 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

49.2.27 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

49.2.28 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

49.2.29 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

49.2.30 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

49.2.31 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

49.2.32 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

49.2.33 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

49.2.34 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

49.2.35 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

49.2.36 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.

49.2.37 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.

49.2.38 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

49.2.39 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

49.2.40 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

49.2.41 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

49.2.42 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

49.2.43 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
Notes	• For STM32F302xC, STM32F303xC, STM32F358xx, STM32F303xE, STM32F398xx and STM32F303x8 two break inputs can be configured.

49.2.44 HAL_TIMEx_RemapConfig

Function Name	**HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap1, uint32_t Remap2)**
Function Description	Configures the TIM1, TIM8, TIM16 and TIM20 Remapping input capabilities.
Parameters	• **htim:** TIM handle. • **Remap1:** specifies the first TIM remapping source. This parameter can be one of the following values: TIM_TIM1_ADC1_NONE: TIM1_ETR is not connected to any ADC1 AWD (analog watchdog) TIM_TIM1_ADC1_AWD1: TIM1_ETR is connected to ADC1 AWD1 TIM_TIM1_ADC1_AWD2: TIM1_ETR is connected to ADC1 AWD2 TIM_TIM1_ADC1_AWD3: TIM1_ETR is connected to ADC1 AWD3 TIM_TIM8_ADC2_NONE: TIM8_ETR is not connected to any ADC2 AWD TIM_TIM8_ADC2_AWD1: TIM8_ETR is connected to ADC2 AWD1 TIM_TIM8_ADC2_AWD2: TIM8_ETR is connected to ADC2 AWD2 TIM_TIM8_ADC2_AWD3: TIM8_ETR is connected to ADC2 AWD3 TIM_TIM16_GPIO: TIM16 TI1 is connected to GPIO TIM_TIM16_RTC: TIM16 TI1 is connected to RTC clock TIM_TIM16_HSE: TIM16 TI1 is connected to HSE/32 TIM_TIM16_MCO: TIM16 TI1 is connected to MCO TIM_TIM20_ADC3_NONE: TIM20_ETR is not connected to any AWD (analog watchdog) TIM_TIM20_ADC3_AWD1: TIM20_ETR is connected to ADC3 AWD1 TIM_TIM20_ADC3_AWD2: TIM20_ETR is connected to ADC3 AWD2 TIM_TIM20_ADC3_AWD3: TIM20_ETR is connected to ADC3 AWD3 • **Remap2:** specifies the second TIMremapping source (if any). This parameter can be one of the following values: TIM_TIM1_ADC4_NONE: TIM1_ETR is not connected to any ADC4 AWD (analog watchdog) TIM_TIM1_ADC4_AWD1: TIM1_ETR is connected to ADC4 AWD1 TIM_TIM1_ADC4_AWD2: TIM1_ETR is connected to ADC4 AWD2 TIM_TIM1_ADC4_AWD3: TIM1_ETR is connected to ADC4 AWD3 TIM_TIM8_ADC3_NONE: TIM8_ETR is not connected to any ADC3 AWD TIM_TIM8_ADC3_AWD1: TIM8_ETR is connected to ADC3 AWD1 TIM_TIM8_ADC3_AWD2: TIM8_ETR is connected to ADC3 AWD2 TIM_TIM8_ADC3_AWD3: TIM8_ETR is connected to

ADC3 AWD3 TIM_TIM16_NONE: Non significant value for TIM16 TIM_TIM20_ADC4_NONE: TIM20_ETR is not connected to any ADC4 AWD TIM_TIM20_ADC4_AWD1: TIM20_ETR is connected to ADC4 AWD1 TIM_TIM20_ADC4_AWD2: TIM20_ETR is connected to ADC4 AWD2 TIM_TIM20_ADC4_AWD3: TIM20_ETR is connected to ADC4 AWD3

Return values
- HAL status

49.2.45 HAL_TIMEx_GroupChannel5

Function Name: **HAL_StatusTypeDef HAL_TIMEx_GroupChannel5 (TIM_HandleTypeDef * htim, uint32_t OCRef)**

Function Description: Group channel 5 and channel 1, 2 or 3.

Parameters
- **htim:** TIM handle.
- **OCRef:** specifies the reference signal(s) the OC5REF is combined with. This parameter can be any combination of the following values: TIM_GROUPCH5_NONE: No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC TIM_GROUPCH5_OC1REFC: OC1REFC is the logical AND of OC1REFC and OC5REF TIM_GROUPCH5_OC2REFC: OC2REFC is the logical AND of OC2REFC and OC5REF TIM_GROUPCH5_OC3REFC: OC3REFC is the logical AND of OC3REFC and OC5REF

Return values
- HAL status

49.2.46 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

49.2.47 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

49.2.48 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

49.3 TIMEx Firmware driver defines

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

49.3.1 TIMEx

TIMEx

TIMEX Break input 2 Enable

TIM_BREAK2_DISABLE

TIM_BREAK2_ENABLE

IS_TIM_BREAK2_STATE

TIM Extended Break Input 2 Polarity

TIM_BREAK2POLARITY_LOW

TIM_BREAK2POLARITY_HIGH

IS_TIM_BREAK2_POLARITY

TIM Extended Break Input Filter

IS_TIM_BREAK_FILTER

TIM Extended Channel

TIM_CHANNEL_1

TIM_CHANNEL_2

TIM_CHANNEL_3

TIM_CHANNEL_4

TIM_CHANNEL_5

TIM_CHANNEL_6

TIM_CHANNEL_ALL

IS_TIM_CHANNELS

IS_TIM_PWMI_CHANNELS

IS_TIM_OPM_CHANNELS

IS_TIM_COMPLEMENTARY_CHANNELS

TIM Extended Clear Input Source

TIM_CLEARINPUTSOURCE_ETR

TIM_CLEARINPUTSOURCE_OCREFCLR

TIM_CLEARINPUTSOURCE_NONE

IS_TIM_CLEARINPUT_SOURCE

TIM Extended Exported Macros

__HAL_TIM_SetCompare

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
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:

- None:

__HAL_TIM_GetCompare

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
  - TIM_CHANNEL_5: get capture/compare 5 register value
  - TIM_CHANNEL_6: get capture/compare 6 register value

Return value:

- None:

Group Channel 5 and Channel 1, 2 or 3

TIM_GROUPCH5_NONE

TIM_GROUPCH5_OC1REFC

TIM_GROUPCH5_OC2REFC

TIM_GROUPCH5_OC3REFC

IS_TIM_GROUPCH5

TIM Extended Master Mode Selection 2 (TRGO2)

TIM_TRGO2_RESET

TIM_TRGO2_ENABLE

TIM_TRGO2_UPDATE

TIM_TRGO2_OC1

TIM_TRGO2_OC1REF

TIM_TRGO2_OC2REF

TIM_TRGO2_OC3REF

TIM_TRGO2_OC4REF

TIM_TRGO2_OC5REF

TIM_TRGO2_OC6REF

TIM_TRGO2_OC4REF_RISINGFALLING

TIM_TRGO2_OC6REF_RISINGFALLING

TIM_TRGO2_OC4REF_RISING_OC6REF_RISING

TIM_TRGO2_OC4REF_RISING_OC6REF_FALLING

TIM_TRGO2_OC5REF_RISING_OC6REF_RISING

TIM_TRGO2_OC5REF_RISING_OC6REF_FALLING

IS_TIM_TRGO2_SOURCE

TIM Extended Output Compare and PWM Modes

TIM_OCMODE_TIMING

TIM_OCMODE_ACTIVE

TIM_OCMODE_INACTIVE

TIM_OCMODE_TOGGLE

TIM_OCMODE_PWM1

TIM_OCMODE_PWM2

TIM_OCMODE_FORCED_ACTIVE

TIM_OCMODE_FORCED_INACTIVE

TIM_OCMODE_RETRIGERRABLE_OPM1

TIM_OCMODE_RETRIGERRABLE_OPM2

TIM_OCMODE_COMBINED_PWM1

TIM_OCMODE_COMBINED_PWM2

TIM_OCMODE_ASSYMETRIC_PWM1

TIM_OCMODE_ASSYMETRIC_PWM2

IS_TIM_PWM_MODE

IS_TIM_OC_MODE

TIM Extended Remapping 1

TIM_TIM1_ADC1_NONE

TIM_TIM1_ADC1_AWD1

TIM_TIM1_ADC1_AWD2

TIM_TIM1_ADC1_AWD3

TIM_TIM8_ADC2_NONE

TIM_TIM8_ADC2_AWD1

TIM_TIM8_ADC2_AWD2

TIM_TIM8_ADC2_AWD3

TIM_TIM16_GPIO

TIM_TIM16_RTC

TIM_TIM16_HSE

TIM_TIM16_MCO

TIM_TIM20_ADC3_NONE

TIM_TIM20_ADC3_AWD1

TIM_TIM20_ADC3_AWD2

TIM_TIM20_ADC3_AWD3

IS_TIM_REMAP

TIM Extended Remapping 2

TIM_TIM1_ADC4_NONE

TIM_TIM1_ADC4_AWD1

TIM_TIM1_ADC4_AWD2

TIM_TIM1_ADC4_AWD3

TIM_TIM8_ADC3_NONE

TIM_TIM8_ADC3_AWD1

TIM_TIM8_ADC3_AWD2

TIM_TIM8_ADC3_AWD3

TIM_TIM16_NONE

TIM_TIM20_ADC4_NONE

TIM_TIM20_ADC4_AWD1

TIM_TIM20_ADC4_AWD2

TIM_TIM20_ADC4_AWD3

IS_TIM_REMAP2

TIM Extended Slave mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

TIM_SLAVEMODE_COMBINED_RESETTRIGGER

IS_TIM_SLAVE_MODE

50 HAL TSC Generic Driver

50.1 TSC Firmware driver registers structures

50.1.1 TSC_InitTypeDef

TSC_InitTypeDef is defined in the stm32f3xx_hal_tsc.h

Data Fields

- ***uint32_t CTPulseHighLength***
- ***uint32_t CTPulseLowLength***
- ***uint32_t SpreadSpectrum***
- ***uint32_t SpreadSpectrumDeviation***
- ***uint32_t SpreadSpectrumPrescaler***
- ***uint32_t PulseGeneratorPrescaler***
- ***uint32_t MaxCountValue***
- ***uint32_t IODefaultMode***
- ***uint32_t SynchroPinPolarity***
- ***uint32_t AcquisitionMode***
- ***uint32_t MaxCountInterrupt***
- ***uint32_t ChannelIOs***
- ***uint32_t ShieldIOs***
- ***uint32_t SamplingIOs***

Field Documentation

- ***uint32_t TSC_InitTypeDef::CTPulseHighLength***
  Charge-transfer high pulse length
- ***uint32_t TSC_InitTypeDef::CTPulseLowLength***
  Charge-transfer low pulse length
- ***uint32_t TSC_InitTypeDef::SpreadSpectrum***
  Spread spectrum activation
- ***uint32_t TSC_InitTypeDef::SpreadSpectrumDeviation***
  Spread spectrum deviation
- ***uint32_t TSC_InitTypeDef::SpreadSpectrumPrescaler***
  Spread spectrum prescaler
- ***uint32_t TSC_InitTypeDef::PulseGeneratorPrescaler***
  Pulse generator prescaler
- ***uint32_t TSC_InitTypeDef::MaxCountValue***
  Max count value
- ***uint32_t TSC_InitTypeDef::IODefaultMode***
  IO default mode
- ***uint32_t TSC_InitTypeDef::SynchroPinPolarity***
  Synchro pin polarity
- ***uint32_t TSC_InitTypeDef::AcquisitionMode***
  Acquisition mode
- ***uint32_t TSC_InitTypeDef::MaxCountInterrupt***
  Max count interrupt activation

- ***uint32_t TSC_InitTypeDef::ChannelIOs***
  Channel IOs mask
- ***uint32_t TSC_InitTypeDef::ShieldIOs***
  Shield IOs mask
- ***uint32_t TSC_InitTypeDef::SamplingIOs***
  Sampling IOs mask

50.1.2 TSC_IOConfigTypeDef

TSC_IOConfigTypeDef is defined in the stm32f3xx_hal_tsc.h

Data Fields

- ***uint32_t ChannelIOs***
- ***uint32_t ShieldIOs***
- ***uint32_t SamplingIOs***

Field Documentation

- ***uint32_t TSC_IOConfigTypeDef::ChannelIOs***
  Channel IOs mask
- ***uint32_t TSC_IOConfigTypeDef::ShieldIOs***
  Shield IOs mask
- ***uint32_t TSC_IOConfigTypeDef::SamplingIOs***
  Sampling IOs mask

50.1.3 TSC_HandleTypeDef

TSC_HandleTypeDef is defined in the stm32f3xx_hal_tsc.h

Data Fields

- ***TSC_TypeDef * Instance***
- ***TSC_InitTypeDef Init***
- ***__IO HAL_TSC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***

Field Documentation

- ***TSC_TypeDef* TSC_HandleTypeDef::Instance***
  Register base address
- ***TSC_InitTypeDef TSC_HandleTypeDef::Init***
  Initialization parameters
- ***__IO HAL_TSC_StateTypeDef TSC_HandleTypeDef::State***
  Peripheral state
- ***HAL_LockTypeDef TSC_HandleTypeDef::Lock***
  Lock feature

50.2 TSC Firmware driver API description

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

50.2.1 TSC specific features

1. Proven and robust surface charge transfer acquisition principle
2. Supports up to 3 capacitive sensing channels per group
3. Capacitive sensing channels can be acquired in parallel offering a very good response time
4. Spread spectrum feature to improve system robustness in noisy environments
5. Full hardware management of the charge transfer acquisition sequence
6. Programmable charge transfer frequency
7. Programmable sampling capacitor I/O pin
8. Programmable channel I/O pin
9. Programmable max count value to avoid long acquisition when a channel is faulty
10. Dedicated end of acquisition and max count error flags with interrupt capability
11. One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
12. Compatible with proximity, touchkey, linear and rotary touch sensor implementation

50.2.2 How to use this driver

1. Enable the TSC interface clock using __TSC_CLK_ENABLE() macro.
2. GPIO pins configuration
   - Enable the clock for the TSC GPIOs using __GPIOx_CLK_ENABLE() macro.
   - Configure the TSC pins used as sampling IOs in alternate function output Open-Drain mode, and TSC pins used as channel/shield IOs in alternate function output Push-Pull mode using HAL_GPIO_Init() function.
   - Configure the alternate function on all the TSC pins using HAL_xxxx() function.
3. Interrupts configuration
   - Configure the NVIC (if the interrupt model is used) using HAL_xxx() function.
4. TSC configuration
   - Configure all TSC parameters and used TSC IOs using HAL_TSC_Init() function.

Acquisition sequence

- Discharge all IOs using HAL_TSC_IODischarge() function.
- Wait a certain time allowing a good discharge of all capacitors. This delay depends of the sampling capacitor and electrodes design.
- Select the channel IOs to be acquired using HAL_TSC_IOConfig() function.
- Launch the acquisition using either HAL_TSC_Start() or HAL_TSC_Start_IT() function. If the synchronized mode is selected, the acquisition will start as soon as the signal is received on the synchro pin.
- Wait the end of acquisition using either HAL_TSC_PollForAcquisition() or HAL_TSC_GetState() function or using WFI instruction for example.
- Check the group acquisition status using HAL_TSC_GroupGetStatus() function.
- Read the acquisition value using HAL_TSC_GroupGetValue() function.

50.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the TSC.
- De-initialize the TSC.
- *HAL_TSC_Init()*
- *HAL_TSC_DeInit()*
- *HAL_TSC_MspInit()*
- *HAL_TSC_MspDeInit()*

50.2.4 IO operation functions

This section provides functions allowing to:

- Start acquisition in polling mode.
- Start acquisition in interrupt mode.
- Stop conversion in polling mode.
- Stop conversion in interrupt mode.
- Get group acquisition status.
- Get group acquisition value.
- *HAL_TSC_Start()*
- *HAL_TSC_Start_IT()*
- *HAL_TSC_Stop()*
- *HAL_TSC_Stop_IT()*
- *HAL_TSC_GroupGetStatus()*
- *HAL_TSC_GroupGetValue()*

50.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure TSC IOs
- Discharge TSC IOs
- *HAL_TSC_IOConfig()*
- *HAL_TSC_IODischarge()*

50.2.6 State functions

This subsection provides functions allowing to

- Get TSC state.
- Poll for acquisition completed.
- Handles TSC interrupt request.
- *HAL_TSC_GetState()*
- *HAL_TSC_PollForAcquisition()*
- *HAL_TSC_IRQHandler()*

50.2.7 HAL_TSC_Init

Function Name	**HAL_StatusTypeDef HAL_TSC_Init (TSC_HandleTypeDef * htsc)**
Function Description	Initializes the TSC peripheral according to the specified parameters in the TSC_InitTypeDef structure.
Parameters	• **htsc:** TSC handle
Return values	• HAL status

50.2.8 HAL_TSC_DeInit

Function Name	**HAL_StatusTypeDef HAL_TSC_DeInit (TSC_HandleTypeDef * htsc)**
Function Description	Deinitializes the TSC peripheral registers to their default reset values.
Parameters	• **htsc:** TSC handle
Return values	• HAL status

50.2.9 HAL_TSC_MspInit

Function Name	**void HAL_TSC_MspInit (TSC_HandleTypeDef * htsc)**
Function Description	Initializes the TSC MSP.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• None

50.2.10 HAL_TSC_MspDeInit

Function Name	**void HAL_TSC_MspDeInit (TSC_HandleTypeDef * htsc)**
Function Description	DeInitializes the TSC MSP.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• None

50.2.11 HAL_TSC_Start

Function Name	**HAL_StatusTypeDef HAL_TSC_Start (TSC_HandleTypeDef * htsc)**
Function Description	Starts the acquisition.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• HAL status

50.2.12 HAL_TSC_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TSC_Start_IT (TSC_HandleTypeDef * htsc)**
Function Description	Enables the interrupt and starts the acquisition.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• HAL status.

50.2.13 HAL_TSC_Stop

Function Name	**HAL_StatusTypeDef HAL_TSC_Stop (TSC_HandleTypeDef * htsc)**
Function Description	Stops the acquisition previously launched in polling mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• HAL status

50.2.14 HAL_TSC_Stop_IT

Function Name	**HAL_StatusTypeDef HAL_TSC_Stop_IT (TSC_HandleTypeDef * htsc)**
Function Description	Stops the acquisition previously launched in interrupt mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• HAL status

50.2.15 HAL_TSC_GroupGetStatus

Function Name	**TSC_GroupStatusTypeDef HAL_TSC_GroupGetStatus (TSC_HandleTypeDef * htsc, uint32_t gx_index)**
Function Description	Gets the acquisition status for a group.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. • **gx_index:** Index of the group
Return values	• Group status

50.2.16 HAL_TSC_GroupGetValue

Function Name	**uint32_t HAL_TSC_GroupGetValue (TSC_HandleTypeDef ***

	htsc, uint32_t gx_index)
Function Description	Gets the acquisition measure for a group.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. • **gx_index:** Index of the group
Return values	• Acquisition measure

50.2.17 HAL_TSC_IOConfig

Function Name	**HAL_StatusTypeDef HAL_TSC_IOConfig (TSC_HandleTypeDef * htsc, TSC_IOConfigTypeDef * config)**
Function Description	Configures TSC IOs.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. • **config:** pointer to the configuration structure.
Return values	• HAL status

50.2.18 HAL_TSC_IODischarge

Function Name	**HAL_StatusTypeDef HAL_TSC_IODischarge (TSC_HandleTypeDef * htsc, uint32_t choice)**
Function Description	Discharge TSC IOs.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC. • **choice:** enable or disable
Return values	• HAL status

50.2.19 HAL_TSC_GetState

Function Name	**HAL_TSC_StateTypeDef HAL_TSC_GetState (TSC_HandleTypeDef * htsc)**
Function Description	Return the TSC state.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• HAL state

50.2.20 HAL_TSC_PollForAcquisition

Function Name	**HAL_StatusTypeDef HAL_TSC_PollForAcquisition (TSC_HandleTypeDef * htsc)**
Function Description	Start acquisition and wait until completion.

Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• HAL state
Notes	• There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.

50.2.21 HAL_TSC_IRQHandler

Function Name	**void HAL_TSC_IRQHandler (TSC_HandleTypeDef * htsc)**
Function Description	Handles TSC interrupt request.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• None

50.2.22 HAL_TSC_ConvCpltCallback

Function Name	**void HAL_TSC_ConvCpltCallback (TSC_HandleTypeDef * htsc)**
Function Description	Acquisition completed callback in non blocking mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• None

50.2.23 HAL_TSC_ErrorCallback

Function Name	**void HAL_TSC_ErrorCallback (TSC_HandleTypeDef * htsc)**
Function Description	Error callback in non blocking mode.
Parameters	• **htsc:** pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
Return values	• None

50.3 TSC Firmware driver defines

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

50.3.1 TSC

TSC

TSC Acquisition mode

TSC_ACQ_MODE_NORMAL

TSC_ACQ_MODE_SYNCHRO

IS_TSC_ACQ_MODE

TSC Charge Transfer Pulse High

TSC_CTPH_1CYCLE

TSC_CTPH_2CYCLES

TSC_CTPH_3CYCLES

TSC_CTPH_4CYCLES

TSC_CTPH_5CYCLES

TSC_CTPH_6CYCLES

TSC_CTPH_7CYCLES

TSC_CTPH_8CYCLES

TSC_CTPH_9CYCLES

TSC_CTPH_10CYCLES

TSC_CTPH_11CYCLES

TSC_CTPH_12CYCLES

TSC_CTPH_13CYCLES

TSC_CTPH_14CYCLES

TSC_CTPH_15CYCLES

TSC_CTPH_16CYCLES

IS_TSC_CTPH

IS_TSC_SS

IS_TSC_SSD

TSC Charge Transfer Pulse Low

TSC_CTPL_1CYCLE

TSC_CTPL_2CYCLES

TSC_CTPL_3CYCLES

TSC_CTPL_4CYCLES

TSC_CTPL_5CYCLES

TSC_CTPL_6CYCLES

TSC_CTPL_7CYCLES

TSC_CTPL_8CYCLES

TSC_CTPL_9CYCLES

TSC_CTPL_10CYCLES

TSC_CTPL_11CYCLES

TSC_CTPL_12CYCLES

TSC_CTPL_13CYCLES

TSC_CTPL_14CYCLES

TSC_CTPL_15CYCLES

TSC_CTPL_16CYCLES

IS_TSC_CTPL

TSC Exported Macros

__HAL_TSC_RESET_HANDLE_STATE

Description:

- Reset TSC handle state.

Parameters:

- __HANDLE__: TSC handle.

Return value:

- None:

__HAL_TSC_ENABLE

Description:

- Enable the TSC peripheral.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None:

__HAL_TSC_DISABLE

Description:

- Disable the TSC peripheral.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None:

__HAL_TSC_START_ACQ

Description:

- Start acquisition.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None:

__HAL_TSC_STOP_ACQ

Description:

- Stop acquisition.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None:

__HAL_TSC_SET_IODEF_OUTPPLOW

Description:

- Set IO default mode to output push-pull low.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None:

__HAL_TSC_SET_IODEF_INFLOAT

Description:

- Set IO default mode to input floating.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None:

__HAL_TSC_SET_SYNC_POL_FALL

Description:

- Set synchronization polarity to falling edge.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None:

__HAL_TSC_SET_SYNC_POL_RISE_HIGH

Description:

- Set synchronization polarity to rising edge and high level.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None:

__HAL_TSC_ENABLE_IT

Description:

- Enable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None:

__HAL_TSC_DISABLE_IT

Description:

- Disable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None:

__HAL_TSC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: TSC Handle
- __INTERRUPT__: TSC interrupt

Return value:

- SET: or RESET

__HAL_TSC_GET_FLAG

Description:

- Get the selected TSC's flag status.

Parameters:

- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:

- SET: or RESET

__HAL_TSC_CLEAR_FLAG

Description:

- Clear the TSC's pending flag.

Parameters:

- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:

- None:

__HAL_TSC_ENABLE_HYSTERESIS

Description:

- Enable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None:

__HAL_TSC_DISABLE_HYSTERESIS

Description:

- Disable schmitt trigger hysteresis on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None:

__HAL_TSC_OPEN_ANALOG_SWITCH

Description:

- Open analog switch on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None:

__HAL_TSC_CLOSE_ANALOG_SWITCH

Description:

- Close analog switch on a group of IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None:

__HAL_TSC_ENABLE_CHANNEL

Description:

- Enable a group of IOs in channel mode.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None:

__HAL_TSC_DISABLE_CHANNEL

Description:

- Disable a group of channel IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None:

__HAL_TSC_ENABLE_SAMPLING

Description:

- Enable a group of IOs in sampling mode.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None:

__HAL_TSC_DISABLE_SAMPLING

Description:

- Disable a group of sampling IOs.

Parameters:

- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:

- None:

__HAL_TSC_ENABLE_GROUP

Description:

- Enable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None:

__HAL_TSC_DISABLE_GROUP

Description:

- Disable acquisition groups.

Parameters:

- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:

- None:

__HAL_TSC_GET_GROUP_STATUS

Description:

- Gets acquisition group status.

Parameters:

- __HANDLE__: TSC Handle
- __GX_INDEX__: Group index

Return value:

- SET: or RESET

TSC Flags Definition

TSC_FLAG_EOA

TSC_FLAG_MCE

TSC groups definition

TSC_NB_OF_GROUPS

TSC_GROUP1

TSC_GROUP2

TSC_GROUP3

TSC_GROUP4

TSC_GROUP5

TSC_GROUP6

TSC_GROUP7

TSC_GROUP8

TSC_ALL_GROUPS

TSC_GROUP1_IDX

TSC_GROUP2_IDX

TSC_GROUP3_IDX

TSC_GROUP4_IDX

TSC_GROUP5_IDX

TSC_GROUP6_IDX

TSC_GROUP7_IDX

TSC_GROUP8_IDX

IS_GROUP_INDEX

TSC_GROUP1_IO1

TSC_GROUP1_IO2

TSC_GROUP1_IO3

TSC_GROUP1_IO4

TSC_GROUP1_ALL_IOS

TSC_GROUP2_IO1

TSC_GROUP2_IO2

TSC_GROUP2_IO3

TSC_GROUP2_IO4

TSC_GROUP2_ALL_IOS

TSC_GROUP3_IO1

TSC_GROUP3_IO2

TSC_GROUP3_IO3

TSC_GROUP3_IO4

TSC_GROUP3_ALL_IOS

TSC_GROUP4_IO1

TSC_GROUP4_IO2

TSC_GROUP4_IO3

TSC_GROUP4_IO4

TSC_GROUP4_ALL_IOS

TSC_GROUP5_IO1

TSC_GROUP5_IO2

TSC_GROUP5_IO3

TSC_GROUP5_IO4

TSC_GROUP5_ALL_IOS

TSC_GROUP6_IO1

TSC_GROUP6_IO2

TSC_GROUP6_IO3

TSC_GROUP6_IO4

TSC_GROUP6_ALL_IOS

TSC_GROUP7_IO1

TSC_GROUP7_IO2

TSC_GROUP7_IO3

TSC_GROUP7_IO4

TSC_GROUP7_ALL_IOS

TSC_GROUP8_IO1

TSC_GROUP8_IO2

TSC_GROUP8_IO3

TSC_GROUP8_IO4

TSC_GROUP8_ALL_IOS

TSC_ALL_GROUPS_ALL_IOS

TSC interrupts definition

TSC_IT_EOA

TSC_IT_MCE

IS_TSC_MCE_IT

TSC I/O default mode definition

TSC_IODEF_OUT_PP_LOW

TSC_IODEF_IN_FLOAT

IS_TSC_IODEF

TSC I/O mode definition

TSC_IOMODE_UNUSED

TSC_IOMODE_CHANNEL

TSC_IOMODE_SHIELD

TSC_IOMODE_SAMPLING

IS_TSC_IOMODE

TSC Max Count Value definition

TSC_MCV_255

TSC_MCV_511

TSC_MCV_1023

TSC_MCV_2047

TSC_MCV_4095

TSC_MCV_8191

TSC_MCV_16383

IS_TSC_MCV

TSC Pulse Generator prescaler definition

TSC_PG_PRESC_DIV1

TSC_PG_PRESC_DIV2

TSC_PG_PRESC_DIV4

TSC_PG_PRESC_DIV8

TSC_PG_PRESC_DIV16

TSC_PG_PRESC_DIV32

TSC_PG_PRESC_DIV64

TSC_PG_PRESC_DIV128

IS_TSC_PG_PRESC

TSC Spread spectrum prescaler definition

TSC_SS_PRESC_DIV1

TSC_SS_PRESC_DIV2

IS_TSC_SS_PRESC

TSC Synchronization pin polarity

TSC_SYNC_POL_FALL

TSC_SYNC_POL_RISE_HIGH

IS_TSC_SYNC_POL

51 HAL UART Generic Driver

51.1 UART Firmware driver registers structures

51.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32f3xx_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***
- ***uint32_t OneBitSampling***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***
 This member configures the UART communication baud rate. The baud rate register is computed using the following formula: If oversampling is 16 or in LIN mode, Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))If oversampling is 8, Baud Rate Register[15:4] = ((2 * PCLKx) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * PCLKx) / ((huart->Init.BaudRate)))[3:0]) >> 1
- ***uint32_t UART_InitTypeDef::WordLength***
 Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UARTEx_Word_Length***
- ***uint32_t UART_InitTypeDef::StopBits***
 Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***
- ***uint32_t UART_InitTypeDef::Parity***
 Specifies the parity mode. This parameter can be a value of ***UART_Parity***
 Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
 Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
 Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***
- ***uint32_t UART_InitTypeDef::OverSampling***
 Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of ***UART_Over_Sampling***
- ***uint32_t UART_InitTypeDef::OneBitSampling***
 Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.

51.1.2 UART_AdvFeatureInitTypeDef

UART_AdvFeatureInitTypeDef is defined in the stm32f3xx_hal_uart.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t AutoBaudRateEnable***
- ***uint32_t AutoBaudRateMode***
- ***uint32_t MSBFirst***

Field Documentation

- ***uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit***
  Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of ***UART_Advanced_Features_Initialization_Type***
- ***uint32_t UART_AdvFeatureInitTypeDef::TxPinLevelInvert***
  Specifies whether the TX pin active level is inverted. This parameter can be a value of ***UART_Tx_Inv***
- ***uint32_t UART_AdvFeatureInitTypeDef::RxPinLevelInvert***
  Specifies whether the RX pin active level is inverted. This parameter can be a value of ***UART_Rx_Inv***
- ***uint32_t UART_AdvFeatureInitTypeDef::DataInvert***
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***UART_Data_Inv***
- ***uint32_t UART_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of ***UART_Rx_Tx_Swap***
- ***uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of ***UART_Overrun_Disable***
- ***uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of ***UART_DMA_Disable_on_Rx_Error***
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable***
  Specifies whether auto Baud rate detection is enabled. This parameter can be a value of ***UART_AutoBaudRate_Enable***
- ***uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode***
  If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of ***UART_AutoBaud_Rate_Mode***
- ***uint32_t UART_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of ***UART_MSB_First***

51.1.3 UART_WakeUpTypeDef

UART_WakeUpTypeDef is defined in the stm32f3xx_hal_uart.h

Data Fields

- ***uint32_t WakeUpEvent***
- ***uint16_t AddressLength***
- ***uint8_t Address***

Field Documentation

- ***uint32_t UART_WakeUpTypeDef::WakeUpEvent***
  Specifies which event will activat the Wakeup from Stop mode flag (WUF). This parameter can be a value of ***UART_WakeUp_from_Stop_Selection***. If set to UART_WAKEUP_ON_ADDRESS, the two other fields below must be filled up.
- ***uint16_t UART_WakeUpTypeDef::AddressLength***
  Specifies whether the address is 4 or 7-bit long. This parameter can be a value of ***UART_WakeUp_Address_Length***
- ***uint8_t UART_WakeUpTypeDef::Address***
  UART/USART node address (7-bit long max)

51.1.4 UART_HandleTypeDef

UART_HandleTypeDef is defined in the stm32f3xx_hal_uart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***UART_InitTypeDef Init***
- ***UART_AdvFeatureInitTypeDef AdvancedInit***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***HAL_UART_StateTypeDef State***
- ***HAL_UART_ErrorTypeDef ErrorCode***

Field Documentation

- ***USART_TypeDef* UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef UART_HandleTypeDef::Init***
  UART communication parameters

- ***UART_AdvFeatureInitTypeDef UART_HandleTypeDef::AdvancedInit***
  UART Advanced Features initialization parameters
- ***uint8_t* UART_HandleTypeDef::pTxBuffPtr***
  Pointer to UART Tx transfer Buffer
- ***uint16_t UART_HandleTypeDef::TxXferSize***
  UART Tx Transfer size
- ***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
- ***uint16_t UART_HandleTypeDef::Mask***
  UART Rx RDR register mask
- ***DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef UART_HandleTypeDef::Lock***
  Locking object
- ***HAL_UART_StateTypeDef UART_HandleTypeDef::State***
  UART communication state
- ***HAL_UART_ErrorTypeDef UART_HandleTypeDef::ErrorCode***
  UART Error code

51.2 UART Firmware driver API description

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

51.2.1 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 these UART pins as alternate function pull-up.
   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 initilalized DMA handle to the UART DMA Tx/Rx handle.
   – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length , Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the huart Init structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart AdvancedInit structure.
5. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
6. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
7. For the UART LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.
8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

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(), HAL_HalfDuplex_Init(), HAL_MultiProcessor_Init(), also configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized 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 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 using HAL_UART_Receive_IT()
- 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

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

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

51.2.2 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) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit), the possible UART frame formats are as listed in *Table 26: "USART frame formats (1 M bit)"* and *Table 27: "USART frame formats (2 M bits)"* :
  - Hardware flow control

- Receiver/transmitter modes
- Over Sampling Method
- One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

Table 29: UART frame formats (1 M bit)

M1, M0 bits	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 |

Table 30: UART frame formats (2 M bits)

M bit	PCE bit	UART frame
10	0	| SB | 7 bit data | STB |
10	1	| SB | 6 bit data | PB | STB |

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init() and HAL_MultiProcessor_Init() API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and multiprocessor configuration procedures (details for the procedures are available in reference manual).

- ***HAL_UART_Init()***
- ***HAL_HalfDuplex_Init()***
- ***HAL_LIN_Init()***
- ***HAL_MultiProcessor_Init()***
- ***HAL_UART_DeInit()***
- ***HAL_UART_MspInit()***
- ***HAL_UART_MspDeInit()***

51.2.3 Peripheral Control functions

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

- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- HAL_MultiProcessor_DisableMuteMode() API disables mute mode
- HAL_MultiProcessor_EnterMuteMode() API enters mute mode
- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- UART_SetConfig() API configures the UART peripheral
- UART_AdvFeatureConfig() API optionally configures the UART advanced features
- UART_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization

- UART_Wakeup_AddressConfig() API configures the wake-up from stop mode parameters
- HAL_HalfDuplex_EnableTransmitter() API disables receiver and enables transmitter
- HAL_HalfDuplex_EnableReceiver() API disables transmitter and enables receiver
- HAL_LIN_SendBreak() API transmits the break characters
- ***HAL_MultiProcessor_EnableMuteMode()***
- ***HAL_MultiProcessor_DisableMuteMode()***
- ***HAL_MultiProcessor_EnterMuteMode()***
- ***UART_SetConfig()***
- ***UART_AdvFeatureConfig()***
- ***UART_CheckIdleState()***
- ***UART_Wakeup_AddressConfig()***
- ***HAL_HalfDuplex_EnableTransmitter()***
- ***HAL_HalfDuplex_EnableReceiver()***
- ***HAL_LIN_SendBreak()***

51.2.4 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 creates the associated handle .
Parameters	• **huart:** UART handle
Return values	• HAL status

51.2.5 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 creates the associated handle .
Parameters	• **huart:** UART handle
Return values	• HAL status

51.2.6 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 creates the associated handle .
Parameters	• **huart:** UART handle • **BreakDetectLength:** specifies the LIN break detection length. This parameter can be one of the following values: UART_LINBREAKDETECTLENGTH_10B: 10-bit break detection UART_LINBREAKDETECTLENGTH_11B: 11-bit

		break detection
Return values	•	HAL status

51.2.7 HAL_MultiProcessor_Init

Function Name	**HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)**
Function Description	Initializes the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.
Parameters	• **huart:** UART handle • **Address:** UART node address (4-, 6-, 7- or 8-bit long) • **WakeUpMethod:** specifies the UART wakeup method. This parameter can be one of the following values: UART_WAKEUPMETHOD_IDLELINE: WakeUp by an idle line detection UART_WAKEUPMETHOD_ADDRESSMARK: WakeUp by an address mark
Return values	• HAL status
Notes	• If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function. • If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

51.2.8 HAL_UART_DeInit

Function Name	**HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)**
Function Description	DeInitializes the UART peripheral.
Parameters	• **huart:** UART handle
Return values	• HAL status

51.2.9 HAL_UART_MspInit

Function Name	**void HAL_UART_MspInit (UART_HandleTypeDef * huart)**
Function Description	UART MSP Init.
Parameters	• **huart:** UART handle
Return values	• None

51.2.10 HAL_UART_MspDeInit

Function Name	**void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)**
Function Description	UART MSP DeInit.
Parameters	• **huart:** UART handle
Return values	• None

51.2.11 HAL_UART_Transmit

Function Name	**HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Send an amount of data in blocking mode.
Parameters	• **huart:** UART handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent • **Timeout:** Timeout duration
Return values	• HAL status

51.2.12 HAL_UART_Receive

Function Name	**HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)**
Function Description	Receive an amount of data in blocking mode.
Parameters	• **huart:** UART handle • **pData:** pointer to data buffer • **Size:** amount of data to be received • **Timeout:** Timeout duration
Return values	• HAL status

51.2.13 HAL_UART_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function Description	Send an amount of data in interrupt mode.
Parameters	• **huart:** UART handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

51.2.14 HAL_UART_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function Description	Receive an amount of data in interrupt mode.
Parameters	• **huart:** UART handle • **pData:** pointer to data buffer • **Size:** amount of data to be received
Return values	• HAL status

51.2.15 HAL_UART_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function Description	Send an amount of data in DMA mode.
Parameters	• **huart:** UART handle • **pData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

51.2.16 HAL_UART_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)**
Function Description	Receive an amount of data in DMA mode.
Parameters	• **huart:** UART handle • **pData:** pointer to data buffer • **Size:** amount of data to be received
Return values	• HAL status
Notes	• When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

51.2.17 HAL_UART_DMAPause

Function Name	**HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)**
Function Description	Pauses the DMA Transfer.
Parameters	• **huart:** UART handle
Return values	• None

51.2.18 HAL_UART_DMAResume

Function Name	**HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)**
Function Description	Resumes the DMA Transfer.
Parameters	• **huart:** UART handle
Return values	• None

51.2.19 HAL_UART_DMAStop

Function Name	**HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)**
Function Description	Stops the DMA Transfer.
Parameters	• **huart:** UART handle
Return values	• None

51.2.20 HAL_UART_IRQHandler

Function Name	**void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)**
Function Description	This function handles UART interrupt request.
Parameters	• **huart:** UART handle
Return values	• None

51.2.21 UART_WaitOnFlagUntilTimeout

Function Name	**HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Timeout)**
Function Description	This function handles UART Communication Timeout.
Parameters	• **huart:** UART handle • **Flag:** specifies the UART flag to check. • **Status:** The new Flag status (SET or RESET). • **Timeout:** Timeout duration
Return values	• HAL status

51.2.22 HAL_UART_TxCpltCallback

Function Name	**void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)**
Function Description	Tx Transfer completed callbacks.

Parameters	• **huart:** UART handle
Return values	• None

51.2.23 HAL_UART_TxHalfCpltCallback

Function Name	**void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)**
Function Description	Tx Half Transfer completed callbacks.
Parameters	• **huart:** UART handle
Return values	• None

51.2.24 HAL_UART_RxCpltCallback

Function Name	**void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)**
Function Description	Rx Transfer completed callbacks.
Parameters	• **huart:** UART handle
Return values	• None

51.2.25 HAL_UART_RxHalfCpltCallback

Function Name	**void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)**
Function Description	Rx Half Transfer completed callbacks.
Parameters	• **huart:** UART handle
Return values	• None

51.2.26 HAL_UART_ErrorCallback

Function Name	**void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)**
Function Description	UART error callbacks.
Parameters	• **huart:** UART handle
Return values	• None

51.2.27 HAL_UART_WakeupCallback

Function Name	**void HAL_UART_WakeupCallback (UART_HandleTypeDef * huart)**

Function Description	UART wakeup from Stop mode callback.
Parameters	• **huart:** UART handle
Return values	• None

51.2.28 HAL_MultiProcessor_EnableMuteMode

Function Name	**HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)**
Function Description	Enable UART in mute mode (doesn't mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called)
Parameters	• **huart:** UART handle
Return values	• HAL status

51.2.29 HAL_MultiProcessor_DisableMuteMode

Function Name	**HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)**
Function Description	Disable UART mute mode (doesn't mean it actually wakes up the software, as it may not have been in mute mode at this very moment).
Parameters	• **huart:** UART handle
Return values	• HAL status

51.2.30 HAL_MultiProcessor_EnterMuteMode

Function Name	**void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)**
Function Description	Enter UART mute mode (means UART actually enters mute mode).
Parameters	• **huart:** UART handle
Return values	• HAL status

51.2.31 UART_SetConfig

Function Name	**HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)**
Function Description	Configure the UART peripheral.
Parameters	• **huart:** UART handle
Return values	• None

51.2.32 UART_AdvFeatureConfig

Function Name	**void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)**
Function Description	Configure the UART peripheral advanced feautures.
Parameters	• **huart:** UART handle
Return values	• None

51.2.33 UART_CheckIdleState

Function Name	**HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)**
Function Description	Check the UART Idle State.
Parameters	• **huart:** UART handle
Return values	• HAL status

51.2.34 UART_Wakeup_AddressConfig

Function Name	**void UART_Wakeup_AddressConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)**
Function Description	Initializes the UART wake-up from stop mode parameters when triggered by address detection.
Parameters	• **huart:** UART handle • **WakeUpSelection:** UART wake up from stop mode parameters
Return values	• HAL status

51.2.35 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:** UART handle
Return values	• HAL status • None

51.2.36 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:** UART handle
Return values	• HAL status

51.2.37 HAL_LIN_SendBreak

Function Name	**HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)**
Function Description	Transmits break characters.
Parameters	• **huart:** UART handle
Return values	• HAL status

51.2.38 HAL_UART_GetState

Function Name	**HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)**
Function Description	return the UART state
Parameters	• **huart:** UART handle
Return values	• HAL state

51.2.39 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

51.3 UART Firmware driver defines

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

51.3.1 UART

UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT

UART_ADVFEATURE_TXINVERT_INIT

UART_ADVFEATURE_RXINVERT_INIT

UART_ADVFEATURE_DATAINVERT_INIT

UART_ADVFEATURE_SWAP_INIT

UART_ADVFEATURE_RXOVERRUNDISABLE_INIT

UART_ADVFEATURE_DMADISABLEONERROR_INIT

UART_ADVFEATURE_AUTOBAUDRATE_INIT

UART_ADVFEATURE_MSBFIRST_INIT

IS_UART_ADVFEATURE_INIT

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE

IS_UART_ADVFEATURE_AUTOBAUDRATE

UART Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT

UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE

UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME

UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME

IS_UART_ADVFEATURE_AUTOBAUDRATEMODE

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRESS_LSB_POS

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRESS_LSB_POS

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LSB_POS

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DATAINV_DISABLE

UART_ADVFEATURE_DATAINV_ENABLE

IS_UART_ADVFEATURE_DATAINV

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA_ENABLEONRXERROR

UART_ADVFEATURE_DMA_DISABLEONRXERROR

IS_UART_ADVFEATURE_DMAONRXERROR

UART DMA Rx

UART_DMA_RX_DISABLE

UART_DMA_RX_ENABLE

IS_UART_DMA_RX

UART DMA Tx

UART_DMA_TX_DISABLE

UART_DMA_TX_ENABLE

IS_UART_DMA_TX

UART DriverEnable Polarity

UART_DE_POLARITY_HIGH

UART_DE_POLARITY_LOW

IS_UART_DE_POLARITY

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE	**Description:** • Reset UART handle state. **Parameters:** • __HANDLE__: UART handle. **Return value:** • None:
__HAL_UART_GET_FLAG	**Description:** • Checks whether the specified UART flag is set or not. **Parameters:** • __HANDLE__: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral (datasheet: up to five USART/UARTs) • __FLAG__: specifies the flag to check. This parameter can be one of the following values: – UART_FLAG_REACK: Receive enable ackowledge flag – UART_FLAG_TEACK: Transmit enable ackowledge flag – UART_FLAG_WUF: Wake up from stop mode flag – UART_FLAG_RWU: Receiver wake up flag (is the UART in mute mode) – UART_FLAG_SBKF: Send Break flag – UART_FLAG_CMF: Character match flag – UART_FLAG_BUSY: Busy flag – UART_FLAG_ABRF: Auto Baud rate detection flag – UART_FLAG_ABRE: Auto Baud rate detection error flag – UART_FLAG_EOBF: End of block flag – UART_FLAG_RTOF: Receiver timeout flag – UART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5) – UART_FLAG_LBD: LIN Break detection flag – UART_FLAG_TXE: Transmit data register empty flag – UART_FLAG_TC: Transmission

Complete flag
  - UART_FLAG_RXNE: Receive data register not empty flag
  - UART_FLAG_IDLE: Idle Line detection flag
  - UART_FLAG_ORE: OverRun Error flag
  - UART_FLAG_NE: Noise Error flag
  - UART_FLAG_FE: Framing Error flag
  - UART_FLAG_PE: Parity Error flag

Return value:

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

__HAL_UART_ENABLE_IT

Description:

- Enables the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral. (datasheet: up to five USART/UARTs)
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART_IT_WUF: Wakeup from stop mode interrupt
  - UART_IT_CM: Character match interrupt
  - UART_IT_CTS: CTS change interrupt
  - UART_IT_LBD: LIN Break detection interrupt
  - UART_IT_TXE: Transmit Data Register empty interrupt
  - UART_IT_TC: Transmission complete interrupt
  - UART_IT_RXNE: Receive Data register not empty interrupt
  - UART_IT_IDLE: Idle line detection interrupt
  - UART_IT_PE: Parity Error interrupt
  - UART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None:

__HAL_UART_DISABLE_IT

Description:

- Disables the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.

This parameter can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral. (datasheet: up to five USART/UARTs)

- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART_IT_WUF: Wakeup from stop mode interrupt
  - UART_IT_CM: Character match interrupt
  - UART_IT_CTS: CTS change interrupt
  - UART_IT_LBD: LIN Break detection interrupt
  - UART_IT_TXE: Transmit Data Register empty interrupt
  - UART_IT_TC: Transmission complete interrupt
  - UART_IT_RXNE: Receive Data register not empty interrupt
  - UART_IT_IDLE: Idle line detection interrupt
  - UART_IT_PE: Parity Error interrupt
  - UART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None:

__HAL_UART_GET_IT

Description:

- Checks whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral. (datasheet: up to five USART/UARTs)
- __IT__: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART_IT_WUF: Wakeup from stop mode interrupt
  - UART_IT_CM: Character match interrupt
  - UART_IT_CTS: CTS change interrupt (not available for UART4 and UART5)
  - UART_IT_LBD: LIN Break detection interrupt
  - UART_IT_TXE: Transmit Data Register empty interrupt
  - UART_IT_TC: Transmission complete interrupt

- UART_IT_RXNE: Receive Data register not empty interrupt
- UART_IT_IDLE: Idle line detection interrupt
- UART_IT_ORE: OverRun Error interrupt
- UART_IT_NE: Noise Error interrupt
- UART_IT_FE: Framing Error interrupt
- UART_IT_PE: Parity Error interrupt

Return value:

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

__HAL_UART_GET_IT_SOURCE

Description:

- Checks whether the specified UART interrupt source is enabled.

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral. (datasheet: up to five USART/UARTs)
- __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_ORE: OverRun Error interrupt
  - UART_IT_NE: Noise Error interrupt
  - UART_IT_FE: Framing Error interrupt
  - UART_IT_PE: Parity Error interrupt

Return value:

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

__HAL_UART_CLEAR_IT

Description:

- Clears the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral. (datasheet: up to five USART/UARTs)

- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - UART_CLEAR_PEF: Parity Error Clear Flag
  - UART_CLEAR_FEF: Framing Error Clear Flag
  - UART_CLEAR_NEF: Noise detected Clear Flag
  - UART_CLEAR_OREF: OverRun Error Clear Flag
  - UART_CLEAR_IDLEF: IDLE line detected Clear Flag
  - UART_CLEAR_TCF: Transmission Complete Clear Flag
  - UART_CLEAR_LBDF: LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF: CTS Interrupt Clear Flag
  - UART_CLEAR_RTOF: Receiver Time Out Clear Flag
  - UART_CLEAR_EOBF: End Of Block Clear Flag
  - UART_CLEAR_CMF: Character Match Clear Flag
  - UART_CLEAR_WUF: Wake Up from stop mode Clear Flag

Return value:

- None:

__HAL_UART_SEND_REQ

Description:

- Set a specific UART request flag.

Parameters:

- __HANDLE__: specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3, 4, 5 to select the USART or UART peripheral. (datasheet: up to five USART/UARTs)
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - UART_AUTOBAUD_REQUEST: Auto-Baud Rate Request
  - UART_SENDBREAK_REQUEST: Send Break Request
  - UART_MUTE_MODE_REQUEST: Mute Mode Request
  - UART_RXDATA_FLUSH_REQUEST:

Receive Data flush Request
  - UART_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None:

__HAL_UART_ENABLE

Description:

- Enable UART.

Parameters:

- __HANDLE__: specifies the UART Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4 or 5 to select the UART peripheral

Return value:

- None:

__HAL_UART_DISABLE

Description:

- Disable UART.

Parameters:

- __HANDLE__: specifies the UART Handle. The Handle Instance can be UARTx where x: 1, 2, 3, 4 or 5 to select the UART peripheral

Return value:

- None:

UART Status Flags

UART_FLAG_REACK

UART_FLAG_TEACK

UART_FLAG_WUF

UART_FLAG_RWU

UART_FLAG_SBKF

UART_FLAG_CMF

UART_FLAG_BUSY

UART_FLAG_ABRF

UART_FLAG_ABRE

UART_FLAG_EOBF

UART_FLAG_RTOF

UART_FLAG_CTS

UART_FLAG_CTSIF

UART_FLAG_LBDF

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 Half Duplex Selection

UART_HALF_DUPLEX_DISABLE

UART_HALF_DUPLEX_ENABLE

IS_UART_HALF_DUPLEX

UART Hardware Flow Control

UART_HWCONTROL_NONE

UART_HWCONTROL_RTS

UART_HWCONTROL_CTS

UART_HWCONTROL_RTS_CTS

IS_UART_HARDWARE_FLOW_CONTROL

UART Interruptions Flag Mask

UART_IT_MASK

UART Interrupts Definition

UART_IT_PE

UART_IT_TXE

UART_IT_TC

UART_IT_RXNE

UART_IT_IDLE

UART_IT_LBD

UART_IT_CTS

UART_IT_CM

UART_IT_WUF

UART_IT_ERR

UART_IT_ORE

UART_IT_NE

UART_IT_FE

UART Interruption Clear Flags

UART_CLEAR_PEF	Parity Error Clear Flag
UART_CLEAR_FEF	Framing Error Clear Flag

UART_CLEAR_NEF	Noise detected Clear Flag
UART_CLEAR_OREF	OverRun Error Clear Flag
UART_CLEAR_IDLEF	IDLE line detected Clear Flag
UART_CLEAR_TCF	Transmission Complete Clear Flag
UART_CLEAR_LBDF	LIN Break Detection Clear Flag
UART_CLEAR_CTSF	CTS Interrupt Clear Flag
UART_CLEAR_RTOF	Receiver Time Out Clear Flag
UART_CLEAR_EOBF	End Of Block Clear Flag
UART_CLEAR_CMF	Character Match Clear Flag
UART_CLEAR_WUF	Wake Up from stop mode Clear Flag

UART Local Interconnection Network mode

UART_LIN_DISABLE

UART_LIN_ENABLE

IS_UART_LIN

UART LIN Break Detection

UART_LINBREAKDETECTLENGTH_10B

UART_LINBREAKDETECTLENGTH_11B

IS_UART_LIN_BREAK_DETECT_LENGTH

UART Transfer Mode

UART_MODE_RX

UART_MODE_TX

UART_MODE_TX_RX

IS_UART_MODE

UART Advanced Feature MSB First

UART_ADVFEATURE_MSBFIRST_DISABLE

UART_ADVFEATURE_MSBFIRST_ENABLE

IS_UART_ADVFEATURE_MSBFIRST

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUTEMODE_DISABLE

UART_ADVFEATURE_MUTEMODE_ENABLE

IS_UART_MUTE_MODE

UART One Bit Sampling Method

UART_ONEBIT_SAMPLING_DISABLED

UART_ONEBIT_SAMPLING_ENABLED

IS_UART_ONEBIT_SAMPLING

UART One Bit sampling

UART_ONE_BIT_SAMPLE_DISABLED

UART_ONE_BIT_SAMPLE_ENABLED

IS_UART_ONEBIT_SAMPLE

UART Advanced Feature Overrun Disable

UART_ADVFEATURE_OVERRUN_ENABLE

UART_ADVFEATURE_OVERRUN_DISABLE

IS_UART_OVERRUN

UART Over Sampling

UART_OVERSAMPLING_16

UART_OVERSAMPLING_8

IS_UART_OVERSAMPLING

UART Parity

UART_PARITY_NONE

UART_PARITY_EVEN

UART_PARITY_ODD

IS_UART_PARITY

UART Private Constants

HAL_UART_TXDMA_TIMEOUTVALUE

UART_CR1_FIELDS

UART Private Macros

__DIV_SAMPLING8

Description:

- BRR division operation to set BRR register in 8-bit oversampling mode.

Parameters:

- _PCLK_: UART clock
- _BAUD_: Baud rate set by the user

Return value:

- Division: result

__DIV_SAMPLING16

Description:

- BRR division operation to set BRR register in 16-bit oversampling mode.

Parameters:

- _PCLK_: UART clock
- _BAUD_: Baud rate set by the user

Return value:

- Division: result

IS_UART_BAUDRATE

Description:

- Check UART Baud rate.

Parameters:

- BAUDRATE: Baudrate specified by the user The maximum Baud Rate is derived from the maximum clock on F3 (i.e. 72 MHz) divided by the smallest oversampling used on the USART (i.e. 8)

Return value:

- Test: result (TRUE or FALSE).

IS_UART_ASSERTIONTIME

Description:

- Check UART assertion time.

Parameters:

- TIME: 5-bit value assertion time

Return value:

- Test: result (TRUE or FALSE).

IS_UART_DEASSERTIONTIME

Description:

- Check UART deassertion time.

Parameters:

- TIME: 5-bit value deassertion time

Return value:

- Test: result (TRUE or FALSE).

UART Receiver TimeOut

UART_RECEIVER_TIMEOUT_DISABLE

UART_RECEIVER_TIMEOUT_ENABLE

IS_UART_RECEIVER_TIMEOUT

UART Request Parameters

UART_AUTOBAUD_REQUEST	Auto-Baud Rate Request
UART_SENDBREAK_REQUEST	Send Break Request
UART_MUTE_MODE_REQUEST	Mute Mode Request
UART_RXDATA_FLUSH_REQUEST	Receive Data flush Request
UART_TXDATA_FLUSH_REQUEST	Transmit data flush Request
IS_UART_REQUEST_PARAMETER	

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXINV_DISABLE

UART_ADVFEATURE_RXINV_ENABLE

IS_UART_ADVFEATURE_RXINV

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWAP_DISABLE

UART_ADVFEATURE_SWAP_ENABLE

IS_UART_ADVFEATURE_SWAP

UART State

UART_STATE_DISABLE

UART_STATE_ENABLE

IS_UART_STATE

UART Number of Stop Bits

UART_STOPBITS_1

UART_STOPBITS_2

IS_UART_STOPBITS

UART Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STOPMODE_DISABLE

UART_ADVFEATURE_STOPMODE_ENABLE

IS_UART_ADVFEATURE_STOPMODE

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VALUE

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXINV_DISABLE

UART_ADVFEATURE_TXINV_ENABLE

IS_UART_ADVFEATURE_TXINV

UART WakeUp Address Length

UART_ADDRESS_DETECT_4B

UART_ADDRESS_DETECT_7B

IS_UART_ADDRESSLENGTH_DETECT

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADDRESS

UART_WAKEUP_ON_STARTBIT

UART_WAKEUP_ON_READDATA_NONEMPTY

IS_UART_WAKEUP_SELECTION

UART WakeUp Methods

UART_WAKEUPMETHOD_IDLELINE

UART_WAKEUPMETHOD_ADDRESSMARK

IS_UART_WAKEUPMETHOD

52 HAL UART Extension Driver

52.1 UARTEx Firmware driver API description

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

52.1.1 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) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit), the possible UART frame formats are as listed in the below tables.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

Table 29: UARTEx frame formats (1 M bit)

M1, M0 bits	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 |

Table 30: UARTEx frame formats (2 M bits)

M bit	PCE bit	UART frame
10	0	| SB | 7 bit data | STB |
10	1	| SB | 6 bit data | PB | STB |

The HAL_RS485Ex_Init() API follows respectively the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

- ***HAL_RS485Ex_Init()***

52.1.2 Peripheral Control functions

This subsection provides an extended function allowing to control the UART.

- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_StopModeWakeUpSourceConfig() configures the address for wake-up from Stop mode based on address match
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- ***HAL_MultiProcessorEx_AddressLength_Set()***
- ***HAL_UARTEx_StopModeWakeUpSourceConfig()***
- ***HAL_UARTEx_EnableStopMode()***
- ***HAL_UARTEx_DisableStopMode()***

52.1.3 HAL_RS485Ex_Init

Function Name	**HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t UART_DEPolarity, uint32_t UART_DEAssertionTime, uint32_t UART_DEDeassertionTime)**
Function Description	Initializes the RS485 Driver enable feature according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	• **huart:** uart handle • **UART_DEPolarity:** select the driver enable polarity This parameter can be one of the following values: UART_DE_POLARITY_HIGH: DE signal is active high UART_DE_POLARITY_LOW: DE signal is active low • **UART_DEAssertionTime:** Driver Enable assertion time 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate) • **UART_DEDeassertionTime:** Driver Enable deassertion time 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).
Return values	• HAL status

52.1.4 HAL_MultiProcessorEx_AddressLength_Set

Function Name	**HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)**
Function Description	By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection.

52.1.5 HAL_UARTEx_StopModeWakeUpSourceConfig

Function Name	**HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)**
Function Description	Set Wakeup from Stop mode interrupt flag selection.
Parameters	• **huart:** uart handle, • **WakeUpSelection:** address match, Start Bit detection or RXNE bit status. This parameter can be one of the following values: UART_WAKEUP_ON_ADDRESS UART_WAKEUP_ON_STARTBIT UART_WAKEUP_ON_READDATA_NONEMPTY
Return values	• HAL status

52.1.6 HAL_UARTEx_EnableStopMode

Function Name	**HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)**
Function Description	Enable UART Stop Mode The UART is able to wake up the MCU from Stop mode as long as UART clock is HSI or LSE.
Parameters	• **huart:** uart handle
Return values	• HAL status

52.1.7 HAL_UARTEx_DisableStopMode

Function Name	**HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)**
Function Description	Disable UART Stop Mode.
Parameters	• **huart:** uart handle
Return values	• HAL status

52.2 UARTEx Firmware driver defines

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

52.2.1 UARTEx

UARTEx

UART Extended Exported Macros

__HAL_UART_GETCLOCKSOURCE

Description:

- Reports the UART clock source.

Parameters:

- __HANDLE__: specifies the UART Handle
- __CLOCKSOURCE__: output variable

Return value:

- UART: clocking source, written in __CLOCKSOURCE__.

__HAL_UART_MASK_COMPUTATION

Description:

- Computes the UART mask to apply to retrieve the received data according to the word length and to the parity bits activation.

Parameters:

- __HANDLE__: specifies the UART Handle

Return value:

- none:

UART Extended Word Length

UART_WORDLENGTH_7B

UART_WORDLENGTH_8B

UART_WORDLENGTH_9B

IS_UART_WORD_LENGTH

53 HAL USART Generic Driver

53.1 USART Firmware driver registers structures

53.1.1 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32f3xx_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: Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))
- ***uint32_t USART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USARTEx_Word_Length***
- ***uint32_t USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***
- ***uint32_t USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_Parity***
  Note: When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t USART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***
- ***uint32_t USART_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***
- ***uint32_t USART_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***
- ***uint32_t USART_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***

53.1.2 USART_HandleTypeDef

USART_HandleTypeDef is defined in the stm32f3xx_hal_usart.h

Data Fields

- ***USART_TypeDef * Instance***
- ***USART_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***HAL_USART_StateTypeDef State***
- ***HAL_USART_ErrorTypeDef 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
- ***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
- ***uint16_t USART_HandleTypeDef::RxXferCount***
  USART Rx Transfer Counter
- ***uint16_t USART_HandleTypeDef::Mask***
  USART Rx RDR register mask
- ***DMA_HandleTypeDef* USART_HandleTypeDef::hdmatx***
  USART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* USART_HandleTypeDef::hdmarx***
  USART Rx DMA Handle parameters
- ***HAL_LockTypeDef USART_HandleTypeDef::Lock***
  Locking object
- ***HAL_USART_StateTypeDef USART_HandleTypeDef::State***
  USART communication state
- ***HAL_USART_ErrorTypeDef USART_HandleTypeDef::ErrorCode***
  USART Error code

53.2 USART Firmware driver API description

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

53.2.1 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 these USART pins as alternate function pull-up.
   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.
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 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 using HAL_USART_Receive_IT()
- 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

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

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

53.2.2 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) or by the M1 and M0 bits (7-bit, 8-bit or 9-bit), the possible USART frame formats are as listed in *Table 26: "USART frame formats (1 M bit)"* and *Table 27: "USART frame formats (2 M bits)"*
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

Table 31: USART frame formats (1 M bit)

M bit	PCE bit	USART 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 |

Table 32: USART frame formats (2 M bits)

M1, M0 bits	PCE bit	USART frame
10	0	| SB | 7 bit data | STB |
10	1	| SB | 6 bit data | PB | STB |

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

- ***HAL_USART_Init()***
- ***HAL_USART_DeInit()***
- ***HAL_USART_MspInit()***
- ***HAL_USART_MspDeInit()***
- ***HAL_USART_CheckIdleState()***

53.2.3 Peripheral Control functions

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

- HAL_USART_GetState() API can be helpful to check in run-time the state of the USART peripheral.
- USART_CheckIdleState() APi ensures that TEACK and/or REACK bits are set after initialization
- *HAL_USART_GetState()*
- *HAL_USART_GetError()*

53.2.4 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:** usart handle
Return values	• HAL status

53.2.5 HAL_USART_DeInit

Function Name	**HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)**
Function Description	DeInitializes the USART peripheral.
Parameters	• **husart:** usart handle
Return values	• HAL status

53.2.6 HAL_USART_MspInit

Function Name	**void HAL_USART_MspInit (USART_HandleTypeDef * husart)**
Function Description	USART MSP Init.
Parameters	• **husart:** usart handle
Return values	• None

53.2.7 HAL_USART_MspDeInit

Function Name	**void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)**
Function Description	USART MSP DeInit.
Parameters	• **husart:** usart handle
Return values	• None

53.2.8 HAL_USART_CheckIdleState

Function Name	**HAL_StatusTypeDef HAL_USART_CheckIdleState (USART_HandleTypeDef * husart)**
Function Description	

53.2.9 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:** USART handle • **pTxData:** pointer to data buffer • **Size:** amount of data to be sent • **Timeout:** Timeout duration
Return values	• HAL status

53.2.10 HAL_USART_Receive

Function Name	**HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	• **husart:** USART handle • **pRxData:** pointer to data buffer • **Size:** amount of data to be received • **Timeout:** Timeout duration
Return values	• HAL status

53.2.11 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 and Receive an amount of data in blocking mode.
Parameters	• **husart:** USART handle • **pTxData:** pointer to TX data buffer • **pRxData:** pointer to RX data buffer • **Size:** amount of data to be sent (same amount to be received) • **Timeout:** Timeout duration
Return values	• HAL status

53.2.12 HAL_USART_Transmit_IT

Function Name	**HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)**
Function Description	Send an amount of data in interrupt mode.
Parameters	• **husart:** USART handle • **pTxData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

53.2.13 HAL_USART_Receive_IT

Function Name	**HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)**
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	• **husart:** usart handle • **pRxData:** pointer to data buffer • **Size:** amount of data to be received
Return values	• HAL status

53.2.14 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 and Receive an amount of data in interrupt mode.
Parameters	• **husart:** USART handle • **pTxData:** pointer to TX data buffer • **pRxData:** pointer to RX data buffer • **Size:** amount of data to be sent (same amount to be received)
Return values	• HAL status

53.2.15 HAL_USART_Transmit_DMA

Function Name	**HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)**
Function Description	Send an amount of data in DMA mode.

Parameters	• **husart:** USART handle • **pTxData:** pointer to data buffer • **Size:** amount of data to be sent
Return values	• HAL status

53.2.16 HAL_USART_Receive_DMA

Function Name	**HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)**
Function Description	Receive an amount of data in DMA mode.
Parameters	• **husart:** USART handle • **pRxData:** pointer to data buffer • **Size:** amount of data to be received
Return values	• HAL status
Notes	• When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position) • The USART DMA transmit channel must be configured in order to generate the clock for the slave.

53.2.17 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:** usart handle • **pTxData:** pointer to TX data buffer • **pRxData:** pointer to RX data buffer • **Size:** amount of data to be received/sent
Return values	• HAL status
Notes	• When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

53.2.18 HAL_USART_DMAPause

Function Name	**HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)**
Function Description	Pauses the DMA Transfer.
Parameters	• **husart:** USART handle
Return values	• None

53.2.19 HAL_USART_DMAResume

Function Name	**HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)**
Function Description	Resumes the DMA Transfer.
Parameters	• **husart:** USART handle
Return values	• None

53.2.20 HAL_USART_DMAStop

Function Name	**HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)**
Function Description	Stops the DMA Transfer.
Parameters	• **husart:** USART handle
Return values	• None

53.2.21 HAL_USART_IRQHandler

Function Name	**void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)**
Function Description	This function handles USART interrupt request.
Parameters	• **husart:** USART handle
Return values	• None

53.2.22 HAL_USART_TxCpltCallback

Function Name	**void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)**
Function Description	Tx Transfer completed callbacks.
Parameters	• **husart:** usart handle
Return values	• None

53.2.23 HAL_USART_TxHalfCpltCallback

Function Name	**void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)**
Function Description	Tx Half Transfer completed callbacks.
Parameters	• **husart:** USART handle
Return values	• None

53.2.24 HAL_USART_RxCpltCallback

Function Name	**void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)**
Function Description	Rx Transfer completed callbacks.
Parameters	• **husart:** USART handle
Return values	• None

53.2.25 HAL_USART_RxHalfCpltCallback

Function Name	**void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)**
Function Description	Rx Half Transfer completed callbacks.
Parameters	• **husart:** usart handle
Return values	• None

53.2.26 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:** usart handle
Return values	• None

53.2.27 HAL_USART_ErrorCallback

Function Name	**void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)**
Function Description	USART error callbacks.
Parameters	• **husart:** usart handle
Return values	• None

53.2.28 HAL_USART_GetState

Function Name	**HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)**
Function Description	return the USART state
Parameters	• **husart:** USART handle
Return values	• HAL state

53.2.29 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

53.3 USART Firmware driver defines

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

53.3.1 USART

USART

USART Clock

USART_CLOCK_DISABLED

USART_CLOCK_ENABLED

IS_USART_CLOCK

USART Clock Phase

USART_PHASE_1EDGE

USART_PHASE_2EDGE

IS_USART_PHASE

USART Clock Polarity

USART_POLARITY_LOW

USART_POLARITY_HIGH

IS_USART_POLARITY

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: USART handle.

Return value:

- None:

__HAL_USART_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the USART Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - USART_FLAG_REACK: Receive enable ackowledge flag
  - USART_FLAG_TEACK: Transmit enable ackowledge flag
  - USART_FLAG_BUSY: Busy flag
  - USART_FLAG_CTS: CTS Change flag
  - USART_FLAG_TXE: Transmit data register empty flag
  - USART_FLAG_TC: Transmission Complete flag
  - USART_FLAG_RXNE: Receive data register not empty flag
  - USART_FLAG_IDLE: Idle Line detection flag
  - USART_FLAG_ORE: OverRun Error flag
  - USART_FLAG_NE: Noise Error flag
  - USART_FLAG_FE: Framing Error flag
  - USART_FLAG_PE: Parity Error flag

Return value:

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

__HAL_USART_ENABLE_IT

Description:

- Enables the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART_IT_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:

- Disables the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - USART_IT_TXE: Transmit Data Register empty interrupt
  - USART_IT_TC: Transmission complete interrupt
  - USART_IT_RXNE: Receive Data register not empty interrupt
  - USART_IT_IDLE: Idle line detection interrupt
  - USART_IT_PE: Parity Error interrupt
  - USART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None:

__HAL_USART_GET_IT

Description:

- Checks whether the specified USART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle
- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_TXE: Transmit Data Register empty interrupt
  - USART_IT_TC: Transmission complete interrupt
  - USART_IT_RXNE: Receive Data register not empty interrupt
  - USART_IT_IDLE: Idle line detection interrupt
  - USART_IT_ORE: OverRun Error interrupt
  - USART_IT_NE: Noise Error interrupt
  - USART_IT_FE: Framing Error interrupt
  - USART_IT_PE: Parity Error interrupt

Return value:

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

__HAL_USART_GET_IT_SOURCE

Description:

- Checks whether the specified USART interrupt source is enabled.

Parameters:

- __HANDLE__: specifies the USART

Handle.

- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_TXE: Transmit Data Register empty interrupt
  - USART_IT_TC: Transmission complete interrupt
  - USART_IT_RXNE: Receive Data register not empty interrupt
  - USART_IT_IDLE: Idle line detection interrupt
  - USART_IT_ORE: OverRun Error interrupt
  - USART_IT_NE: Noise Error interrupt
  - USART_IT_FE: Framing Error interrupt
  - USART_IT_PE: Parity Error interrupt

Return value:

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

__HAL_USART_CLEAR_IT

Description:

- Clears the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - USART_CLEAR_PEF: Parity Error Clear Flag
  - USART_CLEAR_FEF: Framing Error Clear Flag
  - USART_CLEAR_NEF: Noise detected Clear Flag
  - USART_CLEAR_OREF: OverRun Error Clear Flag
  - USART_CLEAR_IDLEF: IDLE line detected Clear Flag
  - USART_CLEAR_TCF: Transmission Complete Clear Flag
  - USART_CLEAR_CTSF: CTS Interrupt Clear Flag

Return value:

- None:

__HAL_USART_SEND_REQ

Description:

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - USART_RXDATA_FLUSH_REQUEST : Receive Data flush Request
  - USART_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None:

__HAL_USART_ENABLE

Description:

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None:

__HAL_USART_DISABLE

Description:

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None:

USART Flags

USART_FLAG_REACK

USART_FLAG_TEACK

USART_FLAG_BUSY

USART_FLAG_CTS

USART_FLAG_CTSIF

USART_FLAG_LBDF

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 interruptions flag mask

USART_IT_MASK

USART Interrupts Definition

USART_IT_PE

USART_IT_TXE

USART_IT_TC

USART_IT_RXNE

USART_IT_IDLE

USART_IT_ERR

USART_IT_ORE

USART_IT_NE

USART_IT_FE

USART Interruption Clear Flags

USART_CLEAR_PEF	Parity Error Clear Flag
USART_CLEAR_FEF	Framing Error Clear Flag
USART_CLEAR_NEF	Noise detected Clear Flag
USART_CLEAR_OREF	OverRun Error Clear Flag
USART_CLEAR_IDLEF	IDLE line detected Clear Flag
USART_CLEAR_TCF	Transmission Complete Clear Flag
USART_CLEAR_CTSF	CTS Interrupt Clear Flag

USART Last Bit

USART_LASTBIT_DISABLE

USART_LASTBIT_ENABLE

IS_USART_LASTBIT

USART Mode

USART_MODE_RX

USART_MODE_TX

USART_MODE_TX_RX

IS_USART_MODE

USART Parity

USART_PARITY_NONE

USART_PARITY_EVEN

USART_PARITY_ODD

IS_USART_PARITY

USART Private Macros

IS_USART_BAUDRATE

Description:

- Check USART Baud rate.

Parameters:

- BAUDRATE: Baudrate specified by the user The maximum Baud Rate is derived from the maximum clock on F3 (i.e. 72 MHz) divided by the smallest oversampling used on the USART (i.e. 8)

Return value:

- Test: result (TRUE or FALSE)

USART Request Parameters

USART_RXDATA_FLUSH_REQUEST Receive Data flush Request

USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

IS_USART_REQUEST_PARAMETER

USART Number of Stop Bits

USART_STOPBITS_1

USART_STOPBITS_0_5

USART_STOPBITS_2

USART_STOPBITS_1_5

IS_USART_STOPBITS

54 HAL USART Extension Driver

54.1 USARTEx Firmware driver defines

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

54.1.1 USARTEx

USARTEx

USART Extended Exported Macros

__HAL_COMP_EXTI_GET_FLAG

Description:

- Checks whether the specified EXTI line flag is set or not.

Parameters:

- __EXTILINE__: specifies the COMP Exti sources to be checked. This parameter can be a value of

Return value:

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

__HAL_COMP_EXTI_CLEAR_FLAG

Description:

- Clear the COMP Exti flags.

Parameters:

- __EXTILINE__: specifies the COMP Exti sources to be cleared. This parameter can be a value of

Return value:

- None.:

__HAL_COMP_EXTI_ENABLE_IT

Description:

- Enable the COMP Exti Line.

Parameters:

- __EXTILINE__: specifies the COMP Exti sources to be enabled. This parameter can be a value of

Return value:

- None.:

__HAL_COMP_EXTI_DISABLE_IT

Description:

- Disable the COMP Exti Line.

Parameters:

- __EXTILINE__: specifies the COMP Exti sources to be disabled. This parameter can be a value of

Return value:

- None.:

__HAL_COMP_EXTI_RISING_IT_ENABLE

Description:

- Enable the Exti Line rising edge trigger.

Parameters:

- __EXTILINE__: specifies the COMP Exti sources to be enabled. This parameter can be a value of

Return value:

- None.:

__HAL_COMP_EXTI_RISING_IT_DISABLE

Description:

- Disable the Exti Line rising edge trigger.

Parameters:

- __EXTILINE__: specifies the COMP Exti sources to be disabled. This parameter can be a value of

Return value:

- None.:

__HAL_COMP_EXTI_FALLING_IT_ENABLE

Description:

- Enable the Exti Line falling edge trigger.

Parameters:

- __EXTILINE__: specifies the COMP Exti sources to be enabled. This parameter can be a value of

Return value:

- None.:

__HAL_COMP_EXTI_FALLING_IT_DISABLE

Description:

- Disable the Exti Line falling edge trigger.

Parameters:

- __EXTILINE__: specifies the COMP Exti sources to be disabled. This parameter can be a value of

Return value:

- None.:

COMP_INIT

Description:

	• Init a comparator instance. **Parameters:** • __HANDLE__: specifies the COMP handle **Return value:** • None.:
COMP_DEINIT	**Description:** • DeInit a comparator instance. **Parameters:** • __HANDLE__: specifies the COMP handle **Return value:** • None.:
COMP_START	**Description:** • Start a comparator instance. **Parameters:** • __HANDLE__: specifies the COMP handle **Return value:** • None.:
COMP_STOP	**Description:** • Stop a comparator instance. **Parameters:** • __HANDLE__: specifies the COMP handle **Return value:** • None.:
COMP_LOCK	**Description:** • Lock a comparator instance. **Parameters:** • __HANDLE__: specifies the COMP handle **Return value:** • None.:
__HAL_COMP_GET_EXTI_LINE	**Description:** • Get the specified EXTI line for a comparator instance. **Parameters:**

- __INSTANCE__: specifies the COMP instance.

Return value:

- value: of

__HAL_USART_GETCLOCKSOURCE

Description:

- Reports the USART clock source.

Parameters:

- __HANDLE__: specifies the USART Handle
- __CLOCKSOURCE__: output variable

Return value:

- the: USART clocking source, written in __CLOCKSOURCE__.

__HAL_USART_MASK_COMPUTATION

Description:

- Computes the USART mask to apply to retrieve the received data according to the word length and to the parity bits activation.

Parameters:

- __HANDLE__: specifies the USART Handle

Return value:

- none:

USART Extended Word Length

USART_WORDLENGTH_7B

USART_WORDLENGTH_8B

USART_WORDLENGTH_9B

IS_USART_WORD_LENGTH

55 HAL WWDG Generic Driver

55.1 WWDG Firmware driver registers structures

55.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32f3xx_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

55.1.2 WWDG_HandleTypeDef

WWDG_HandleTypeDef is defined in the stm32f3xx_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

55.2 WWDG Firmware driver API description

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

55.2.1 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) = PCLK1 / (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 @42 MHz(PCLK1): ~97.5 us / ~49.9 ms

55.2.2 How to use this driver

- Enable WWDG APB1 clock using __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

55.2.3 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()***

55.2.4 Peripheral State functions

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

- ***HAL_WWDG_GetState()***

55.2.5 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

55.2.6 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

55.2.7 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

55.2.8 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

55.2.9 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

55.2.10 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

55.2.11 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

55.2.12 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

55.2.13 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 using __HAL_WWDG_ENABLE_IT() macro. 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.

55.2.14 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

55.2.15 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

55.3 WWDG Firmware driver defines

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

55.3.1 WWDG

WWDG

WWDG registers bit address in the alias region

CFR_BASE

WWDG Counter

IS_WWDG_COUNTER

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_ENABLE_IT	**Description:** • Enables the WWDG early wakeup interrupt. **Parameters:** • __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_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:

WWDG Flag definition

WWDG_FLAG_EWIF Early wakeup interrupt flag

IS_WWDG_FLAG

WWDG Interrupt definition

WWDG_IT_EWI	Early wakeup interrupt
IS_WWDG_IT	

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
IS_WWDG_PRESCALER	

WWDG Window

IS_WWDG_WINDOW

56 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 STM32F3 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32F3 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 *Table 5: "List of 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: stm32f3xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration, …)

A template is provided in the HAL driver folders (stm32f3xx_hal_conf_template.h).

Which header files should I include in my application to use the HAL drivers?

Only stm32f3xx_hal.h file has to be included.

What is the difference between stm32f3xx_hal_ppp.c/.h and stm32f3xx_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 (stm32f3xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32f3xx_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_stm32f3xx.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 stm32f3xx_hal_msp.c. A template is provided in the HAL driver folders (stm32f3xx_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 stm32f3xx_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 stm32f3xx_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 stm32f3xx_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.

57 Revision history

Table 33: Document revision history

Date	Revision	Changes
16-Feb-2015	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

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

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

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

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

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

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

© 2015 STMicroelectronics – All rights reserved