

## Description of STM32F4xx HAL drivers

### Introduction

STM32Cube™ 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 STM32CubeF4 for STM32F4 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  - All embedded software utilities coming with a full set of examples.

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

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

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

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

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

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

This user manual is structured as follows:

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



## Contents

<b>1</b>	<b>Acronyms and definitions.....</b>	<b>26</b>
<b>2</b>	<b>Overview of HAL drivers .....</b>	<b>28</b>
2.1	HAL and user application files.....	28
2.1.1	HAL driver files .....	28
2.1.2	User-application files .....	29
2.2	HAL data structures .....	31
2.2.1	Peripheral handle structures .....	31
2.2.2	Initialization and configuration structure .....	32
2.2.3	Specific process structures .....	33
2.3	API classification .....	33
2.4	Devices supported by HAL drivers .....	34
2.5	HAL drivers rules.....	37
2.5.1	HAL API naming rules .....	37
2.5.2	HAL general naming rules .....	38
2.5.3	HAL interrupt handler and callback functions.....	39
2.6	HAL generic APIs.....	39
2.7	HAL extension APIs .....	41
2.7.1	HAL extension model overview .....	41
2.7.2	HAL extension model cases .....	41
2.8	File inclusion model.....	43
2.9	HAL common resources.....	44
2.10	HAL configuration.....	45
2.11	HAL system peripheral handling .....	46
2.11.1	Clock.....	46
2.11.2	GPIOs.....	47
2.11.3	Cortex NVIC and SysTick timer.....	49
2.11.4	PWR .....	49
2.11.5	EXTI.....	50
2.11.6	DMA.....	50
2.12	How to use HAL drivers .....	52
2.12.1	HAL usage models .....	52
2.12.2	HAL initialization .....	53
2.12.3	HAL IO operation process .....	55
2.12.4	Timeout and error management.....	58

<b>3 HAL common driver .....</b>	<b>62</b>
3.1 HAL Firmware driver API description .....	62
3.1.1 How to use this driver .....	62
3.1.2 Initialization and de-initialization functions .....	62
3.1.3 HAL Control functions.....	62
3.1.4 Initialization and de-initialization Functions .....	63
3.1.5 HAL Control functions.....	64
3.2 HAL Firmware driver defines.....	69
3.2.1 HAL.....	69
<b>4 HAL ADC Generic Driver.....</b>	<b>70</b>
4.1 ADC Firmware driver registers structures .....	70
4.1.1 ADC_HandleTypeDef .....	70
4.1.2 ADC_InitTypeDef.....	70
4.1.3 ADC_ChannelConfTypeDef .....	72
4.1.4 ADC_AnalogWDGConfTypeDef.....	72
4.1.5 ADC_Common_TypeDef.....	73
4.1.6 ADC_TypeDef .....	73
4.2 ADC Firmware driver API description.....	75
4.2.1 ADC Peripheral features.....	75
4.2.2 How to use this driver .....	75
4.2.3 Initialization and de-initialization functions .....	77
4.2.4 IO operation functions .....	77
4.2.5 Peripheral Control functions .....	77
4.2.6 Peripheral State and errors functions.....	78
4.2.7 Initialization and de-initialization functions .....	78
4.2.8 IO operation functions .....	79
4.2.9 Peripheral Control functions .....	84
4.2.10 ADC Peripheral State functions.....	85
4.3 ADC Firmware driver defines .....	86
4.3.1 ADC .....	86
<b>5 HAL ADC Extension Driver .....</b>	<b>95</b>
5.1 ADCEx Firmware driver registers structures .....	95
5.1.1 ADC_InjectionConfTypeDef .....	95
5.1.2 ADC_MultiModeTypeDef.....	96
5.2 ADCEx Firmware driver API description .....	96
5.2.1 How to use this driver .....	96
5.2.2 Extended features functions.....	98

Contents	UM1725
5.2.3    Extended features functions .....	98
5.3    ADCEx Firmware driver defines .....	103
5.3.1    ADCEx .....	103
<b>6    HAL CAN Generic Driver.....</b>	<b>110</b>
6.1    CAN Firmware driver registers structures .....	110
6.1.1    CAN_HandleTypeDef .....	110
6.1.2    CAN_InitTypeDef .....	110
6.1.3    CAN_FilterConfTypeDef .....	111
6.1.4    CAN_FIFOMailBox_TypeDef .....	112
6.1.5    CAN_FilterRegister_TypeDef .....	113
6.1.6    CAN_TxMailBox_TypeDef .....	113
6.1.7    CAN_TypeDef .....	114
6.2    CAN Firmware driver API description.....	115
6.2.1    How to use this driver .....	115
6.2.2    Initialization and de-initialization functions .....	116
6.2.3    IO operation functions .....	117
6.2.4    Peripheral State and Error functions .....	117
6.2.5    Initialization and de-initialization functions .....	117
6.2.6    IO operation functions .....	119
6.2.7    Peripheral State and Error functions .....	123
6.3    CAN Firmware driver defines .....	123
6.3.1    CAN .....	123
<b>7    HAL CORTEX Generic Driver.....</b>	<b>133</b>
7.1    CORTEX Firmware driver API description .....	133
7.1.1    How to use this driver .....	133
7.1.2    Initialization and de-initialization functions .....	134
7.1.3    Peripheral Control functions .....	134
7.1.4    Initialization and de-initialization functions .....	134
7.1.5    Peripheral Control functions .....	137
7.2    CORTEX Firmware driver defines .....	140
7.2.1    CORTEX .....	140
<b>8    HAL CRC Generic Driver.....</b>	<b>142</b>
8.1    CRC Firmware driver registers structures .....	142
8.1.1    CRC_HandleTypeDef .....	142
8.1.2    CRC_TypeDef .....	142
8.2    CRC Firmware driver API description .....	143
8.2.1    How to use this driver .....	143

8.2.2	Initialization and de-initialization functions .....	143
8.2.3	Peripheral Control functions .....	143
8.2.4	Peripheral State functions .....	143
8.2.5	Initialization and de-initialization functions .....	144
8.2.6	Peripheral Control functions .....	145
8.2.7	Peripheral State functions .....	146
8.3	CRC Firmware driver defines .....	146
8.3.1	CRC .....	146
<b>9</b>	<b>HAL CRYP Generic Driver.....</b>	<b>147</b>
9.1	CRYP Firmware driver registers structures .....	147
9.1.1	CRYP_HandleTypeDef.....	147
9.1.2	CRYP_InitTypeDef .....	148
9.1.3	CRYP_TypeDef.....	148
9.2	CRYP Firmware driver API description .....	151
9.2.1	How to use this driver .....	151
9.2.2	Initialization and de-initialization functions .....	152
9.2.3	AES processing functions .....	152
9.2.4	DES processing functions .....	153
9.2.5	TDES processing functions .....	153
9.2.6	DMA callback functions .....	153
9.2.7	CRYP IRQ handler management.....	154
9.2.8	Peripheral State functions .....	154
9.2.9	Initialization and de-initialization functions .....	154
9.2.10	AES processing functions .....	155
9.2.11	DES processing functions .....	164
9.2.12	TDES processing functions .....	169
9.2.13	DMA callback functions .....	174
9.2.14	CRYP IRQ handler management.....	175
9.2.15	Peripheral State functions .....	176
9.3	CRYP Firmware driver defines .....	176
9.3.1	CRYP .....	176
<b>10</b>	<b>HAL DAC Generic Driver.....</b>	<b>180</b>
10.1	DAC Firmware driver registers structures .....	180
10.1.1	DAC_HandleTypeDef.....	180
10.1.2	DAC_ChannelConfTypeDef .....	180
10.1.3	DAC_TypeDef .....	181
10.2	DAC Firmware driver API description.....	182

Contents	UM1725
10.2.1 DAC Peripheral features .....	182
10.2.2 How to use this driver .....	183
10.2.3 Initialization and de-initialization functions .....	184
10.2.4 IO operation functions .....	184
10.2.5 Peripheral Control functions .....	184
10.2.6 Peripheral State and Errors functions .....	185
10.2.7 Initialization and de-initialization functions .....	185
10.2.8 IO operation functions .....	186
10.2.9 Peripheral Control functions .....	190
10.2.10 Peripheral State and Errors functions .....	191
10.3 DAC Firmware driver defines .....	192
10.3.1 DAC .....	192
<b>11 HAL DAC Extension Driver .....</b>	<b>196</b>
11.1 DACEx Firmware driver API description .....	196
11.1.1 How to use this driver .....	196
11.1.2 Extended features functions .....	196
11.1.3 Extended features functions .....	196
11.2 DACEx Firmware driver defines .....	199
11.2.1 DACEx .....	199
<b>12 HAL DMA Generic Driver .....</b>	<b>203</b>
12.1 DMA Firmware driver registers structures .....	203
12.1.1 DMA_HandleTypeDef .....	203
12.1.2 DMA_InitTypeDef .....	204
12.1.3 DMA_Stream_TypeDef .....	205
12.1.4 DMA_TypeDef .....	205
12.2 DMA Firmware driver API description .....	206
12.2.1 How to use this driver .....	206
12.2.2 Initialization and de-initialization functions .....	207
12.2.3 IO operation functions .....	207
12.2.4 State and Errors functions .....	208
12.2.5 Initialization and de-initialization functions .....	208
12.2.6 I/O operation functions .....	209
12.2.7 Peripheral State functions .....	211
12.3 DMA Firmware driver defines .....	212
12.3.1 DMA .....	212
<b>13 HAL DMA Extension Driver .....</b>	<b>220</b>
13.1 DMAEx Firmware driver API description .....	220

13.1.1	How to use this driver .....	220
13.1.2	Extended features functions .....	220
13.1.3	Extended features functions .....	220
13.2	DMAEx Firmware driver defines.....	222
13.2.1	DMAEx.....	222
<b>14</b>	<b>HAL DMA2D Generic Driver .....</b>	<b>223</b>
14.1	DMA2D Firmware driver registers structures .....	223
14.1.1	DMA2D_HandleTypeDef .....	223
14.1.2	DMA2D_InitTypeDef.....	223
14.1.3	DMA2D_LayerCfgTypeDef.....	224
14.1.4	DMA2D_ColorTypeDef.....	224
14.1.5	DMA2D_CLUTCfgTypeDef .....	225
14.1.6	DMA2D_TypeDef .....	225
14.2	DMA2D Firmware driver API description.....	227
14.2.1	How to use this driver .....	227
14.2.2	Initialization and Configuration functions .....	228
14.2.3	IO operation functions .....	228
14.2.4	Peripheral Control functions .....	229
14.2.5	Peripheral State and Errors functions .....	229
14.2.6	Initialization and Configuration functions .....	229
14.2.7	IO operation functions .....	231
14.2.8	Peripheral Control functions .....	235
14.2.9	Peripheral State functions .....	237
14.3	DMA2D Firmware driver defines .....	238
14.3.1	DMA2D .....	238
<b>15</b>	<b>HAL DCMI Generic Driver .....</b>	<b>243</b>
15.1	DCMI Firmware driver registers structures.....	243
15.1.1	DCMI_HandleTypeDef .....	243
15.1.2	DCMI_InitTypeDef .....	243
15.1.3	DCMI_CodesInitTypeDef.....	244
15.1.4	DCMI_TypeDef.....	245
15.2	DCMI Firmware driver API description .....	246
15.2.1	How to use this driver .....	246
15.2.2	Initialization and Configuration functions .....	246
15.2.3	IO operation functions .....	247
15.2.4	Peripheral Control functions .....	247
15.2.5	Peripheral State and Errors functions .....	247

15.2.6	Initialization and Configuration functions .....	247
15.2.7	IO operation functions .....	249
15.2.8	Peripheral Control functions .....	251
15.2.9	Peripheral State functions .....	253
15.3	DCMI Firmware driver defines.....	253
15.3.1	DCMI.....	253
<b>16</b>	<b>HAL ETHERNET Generic Driver .....</b>	<b>258</b>
16.1	ETH Firmware driver registers structures.....	258
16.1.1	ETH_HandleTypeDef .....	258
16.1.2	ETH_InitTypeDef.....	258
16.1.3	ETH_MACInitTypeDef .....	259
16.1.4	ETH_DMADescTypeDef.....	262
16.1.5	ETH_DMAInitTypeDef .....	263
16.1.6	ETH_DMARxFrameInfos.....	264
16.1.7	ETH_TypeDef.....	265
16.2	ETH Firmware driver API description .....	267
16.2.1	How to use this driver .....	267
16.2.2	Initialization and de-initialization functions .....	268
16.2.3	IO operation functions .....	268
16.2.4	Peripheral Control functions .....	269
16.2.5	Peripheral State functions .....	269
16.2.6	Initialization and de-initialization functions .....	269
16.2.7	IO operation functions .....	271
16.2.8	Peripheral Control functions .....	275
16.2.9	Peripheral State functions .....	276
16.3	ETH Firmware driver defines.....	277
16.3.1	ETH.....	277
<b>17</b>	<b>HAL FLASH Generic Driver.....</b>	<b>311</b>
17.1	FLASH Firmware driver registers structures .....	311
17.1.1	FLASH_ProcessTypeDef .....	311
17.1.2	FLASH_TypeDef .....	311
17.2	FLASH Firmware driver API description .....	312
17.2.1	FLASH peripheral features .....	312
17.2.2	How to use this driver .....	312
17.2.3	Programming operation functions .....	313
17.2.4	Peripheral Control functions .....	313
17.2.5	Peripheral Errors functions .....	313

17.2.6	Programming operation functions .....	313
17.2.7	Peripheral Control functions .....	315
17.2.8	Peripheral State and Errors functions .....	317
17.3	FLASH Firmware driver defines .....	318
17.3.1	FLASH .....	318
<b>18</b>	<b>HAL FLASH Extension Driver .....</b>	<b>323</b>
18.1	FLASHEx Firmware driver registers structures .....	323
18.1.1	FLASH_EraseInitTypeDef .....	323
18.1.2	FLASH_OBProgramInitTypeDef .....	323
18.1.3	FLASH_AdvOBProgramInitTypeDef .....	324
18.2	FLASHEx Firmware driver API description.....	325
18.2.1	Flash Extension features .....	325
18.2.2	How to use this driver .....	325
18.2.3	Extended programming operation functions .....	326
18.2.4	Extended IO operation functions .....	326
18.3	FLASHEx Firmware driver defines .....	330
18.3.1	FLASHEx .....	330
<b>19</b>	<b>HAL GPIO Generic Driver.....</b>	<b>340</b>
19.1	GPIO Firmware driver registers structures .....	340
19.1.1	GPIO_InitTypeDef .....	340
19.1.2	GPIO_TypeDef .....	340
19.2	GPIO Firmware driver API description .....	341
19.2.1	GPIO Peripheral features .....	341
19.2.2	How to use this driver .....	342
19.2.3	Initialization and de-initialization functions .....	342
19.2.4	IO operation functions .....	342
19.2.5	Initialization and de-initialization functions .....	343
19.2.6	IO operation functions .....	344
19.3	GPIO Firmware driver defines.....	346
19.3.1	GPIO .....	346
<b>20</b>	<b>HAL HASH Generic Driver .....</b>	<b>354</b>
20.1	HASH Firmware driver registers structures .....	354
20.1.1	HASH_HandleTypeDef.....	354
20.1.2	HASH_InitTypeDef .....	355
20.1.3	HASH_DIGEST_TypeDef.....	355
20.1.4	HASH_TypeDef .....	355

Contents	UM1725
20.2 HASH Firmware driver API description .....	356
20.2.1 How to use this driver .....	356
20.2.2 Initialization and de-initialization functions .....	357
20.2.3 HASH processing using polling mode functions .....	357
20.2.4 HASH processing using interrupt mode functions .....	358
20.2.5 HASH processing using DMA mode functions .....	358
20.2.6 HMAC processing using polling mode functions .....	358
20.2.7 HMAC processing using DMA mode functions .....	358
20.2.8 Peripheral State functions .....	358
20.2.9 Initialization and de-initialization functions .....	359
20.2.10 HASH processing functions using polling mode .....	361
20.2.11 HASH processing functions using interrupt mode .....	363
20.2.12 HASH processing functions using DMA mode .....	364
20.2.13 HASH-MAC (HMAC) processing functions using polling mode .....	366
20.2.14 HASH-MAC (HMAC) processing functions using DMA mode.....	367
20.2.15 Peripheral State functions .....	368
20.3 HASH Firmware driver defines.....	369
20.3.1 HASH.....	369
<b>21 HAL HASH Extension Driver.....</b>	<b>371</b>
21.1 HASHEx Firmware driver API description .....	371
21.1.1 How to use this driver .....	371
21.1.2 HASH processing using polling mode functions .....	372
21.1.3 HMAC processing using polling mode functions .....	372
21.1.4 HASH processing using interrupt functions.....	372
21.1.5 HASH processing using DMA functions .....	372
21.1.6 HMAC processing using DMA functions .....	373
21.1.7 HASH processing functions.....	373
21.1.8 HMAC processing functions using polling mode .....	375
21.1.9 HASH processing functions using interrupt mode .....	376
21.1.10 HASH processing functions using DMA mode .....	377
21.1.11 HMAC processing functions using DMA mode .....	379
21.2 HASHEx Firmware driver defines.....	380
21.2.1 HASHEx .....	380
<b>22 HAL HCD Generic Driver.....</b>	<b>381</b>
22.1 HCD Firmware driver registers structures .....	381
22.1.1 HCD_HandleTypeDef.....	381
22.2 HCD Firmware driver API description .....	381

22.2.1	How to use this driver .....	381
22.2.2	Initialization and de-initialization functions .....	382
22.2.3	IO operation functions .....	382
22.2.4	Peripheral Control functions .....	382
22.2.5	Peripheral State functions .....	382
22.2.6	Initialization and de-initialization functions .....	383
22.2.7	IO operation functions .....	385
22.2.8	Peripheral Control functions .....	387
22.2.9	Peripheral State functions .....	388
22.3	HCD Firmware driver defines .....	391
22.3.1	HCD .....	391
<b>23</b>	<b>HAL I2C Generic Driver .....</b>	<b>392</b>
23.1	I2C Firmware driver registers structures .....	392
23.1.1	I2C_HandleTypeDef .....	392
23.1.2	I2C_InitTypeDef .....	392
23.1.3	I2C_TypeDef .....	393
23.2	I2C Firmware driver API description.....	394
23.2.1	How to use this driver .....	394
23.2.2	Initialization and de-initialization functions .....	397
23.2.3	IO operation functions .....	397
23.2.4	Peripheral State and Errors functions .....	399
23.2.5	Initialization and de-initialization functions .....	399
23.2.6	IO operation functions .....	400
23.2.7	Peripheral State and Errors functions .....	412
23.3	I2C Firmware driver defines .....	413
23.3.1	I2C .....	413
<b>24</b>	<b>HAL I2C Extension Driver .....</b>	<b>417</b>
24.1	I2CEx Firmware driver API description .....	417
24.1.1	I2C peripheral extension features .....	417
24.1.2	How to use this driver .....	417
24.1.3	Extension features functions .....	417
24.1.4	Extension features functions .....	417
24.2	I2CEx Firmware driver defines .....	418
24.2.1	I2CEx .....	418
<b>25</b>	<b>HAL I2S Generic Driver .....</b>	<b>419</b>
25.1	I2S Firmware driver registers structures .....	419
25.1.1	I2S_HandleTypeDef .....	419

Contents	UM1725
25.1.2 I2S_InitTypeDef .....	419
25.1.3 SPI_TypeDef .....	420
25.2 I2S Firmware driver API description .....	421
25.2.1 How to use this driver .....	421
25.2.2 Initialization and de-initialization functions .....	423
25.2.3 IO operation functions .....	423
25.2.4 Peripheral State and Errors functions .....	424
25.2.5 Initialization and de-initialization functions .....	424
25.2.6 IO operation functions .....	426
25.2.7 Peripheral State and Errors functions .....	432
25.3 I2S Firmware driver defines .....	433
25.3.1 I2S .....	433
<b>26 HAL I2S Extension Driver .....</b>	<b>438</b>
26.1 I2SEEx Firmware driver API description .....	438
26.1.1 I2S Extension features .....	438
26.1.2 How to use this driver .....	438
26.1.3 Extension features Functions .....	439
26.1.4 Extension features functions .....	439
26.2 I2SEEx Firmware driver defines .....	441
26.2.1 I2SEEx .....	441
<b>27 HAL IRDA Generic Driver .....</b>	<b>442</b>
27.1 IRDA Firmware driver registers structures .....	442
27.1.1 IRDA_HandleTypeDef .....	442
27.1.2 IRDA_InitTypeDef .....	442
27.1.3 USART_TypeDef .....	443
27.2 IRDA Firmware driver API description .....	444
27.2.1 How to use this driver .....	444
27.2.2 Initialization and Configuration functions .....	445
27.2.3 IO operation functions .....	446
27.2.4 Peripheral State and Errors functions .....	446
27.2.5 IrDA Initialization and de-initialization functions .....	447
27.2.6 IO operation functions .....	448
27.2.7 Peripheral State and Errors functions .....	452
27.3 IRDA Firmware driver defines .....	453
27.3.1 IRDA .....	453
<b>28 HAL IWDG Generic Driver .....</b>	<b>456</b>
28.1 IWDG Firmware driver registers structures .....	456

28.1.1	IWDG_HandleTypeDef .....	456
28.1.2	IWDG_InitTypeDef .....	456
28.1.3	IWDG_TypeDef .....	457
28.2	IWDG Firmware driver API description .....	457
28.2.1	IWDG Generic features .....	457
28.2.2	How to use this driver .....	458
28.2.3	Initialization and de-initialization functions .....	458
28.2.4	IO operation functions .....	458
28.2.5	Peripheral State functions .....	458
28.2.6	Initialization and de-initialization functions .....	459
28.2.7	IO operation functions .....	459
28.2.8	Peripheral State functions .....	460
28.3	IWDG Firmware driver defines .....	461
28.3.1	IWDG .....	461
<b>29</b>	<b>HAL LTDC Generic Driver .....</b>	<b>463</b>
29.1	LTDC Firmware driver registers structures .....	463
29.1.1	LTDC_HandleTypeDef .....	463
29.1.2	LTDC_InitTypeDef .....	463
29.1.3	LTDC_ColorTypeDef .....	464
29.1.4	LTDC_LayerCfgTypeDef .....	465
29.1.5	LTDC_Layer_TypeDef .....	466
29.1.6	LTDC_TypeDef .....	467
29.2	LTDC Firmware driver API description .....	469
29.2.1	How to use this driver .....	469
29.2.2	Initialization and Configuration functions .....	469
29.2.3	IO operation functions .....	470
29.2.4	Peripheral Control functions .....	470
29.2.5	Peripheral State and Errors functions .....	470
29.2.6	Initialization and Configuration functions .....	471
29.2.7	IO operation functions .....	472
29.2.8	Peripheral Control functions .....	473
29.2.9	Peripheral State and Errors functions .....	479
29.3	LTDC Firmware driver defines .....	480
29.3.1	LTDC .....	480
<b>30</b>	<b>HAL NAND Generic Driver .....</b>	<b>484</b>
30.1	NAND Firmware driver registers structures .....	484
30.1.1	NAND_HandleTypeDef .....	484

Contents	UM1725
30.1.2 NAND_AddressTypeDef .....	484
30.1.3 NAND_IDTypeDef .....	485
30.2 NAND Firmware driver API description .....	485
30.2.1 How to use this driver .....	485
30.2.2 NAND Initialization and de-initialization functions .....	486
30.2.3 NAND Input and Output functions .....	486
30.2.4 NAND Control functions .....	486
30.2.5 NAND State functions .....	486
30.2.6 Initialization and de-initialization functions .....	487
30.2.7 Input and Output functions .....	489
30.2.8 Control functions .....	493
30.2.9 State functions .....	494
30.3 NAND Firmware driver defines .....	494
30.3.1 NAND .....	494
<b>31 HAL NOR Generic Driver .....</b>	<b>496</b>
31.1 NOR Firmware driver registers structures .....	496
31.1.1 NOR_HandleTypeDef .....	496
31.1.2 NOR_CFITTypeDef .....	496
31.1.3 NOR_IDTypeDef .....	497
31.2 NOR Firmware driver API description .....	497
31.2.1 How to use this driver .....	497
31.2.2 NOR Initialization and de_initialization functions .....	498
31.2.3 NOR Input and Output functions .....	498
31.2.4 NOR Control functions .....	498
31.2.5 NOR State functions .....	498
31.2.6 Initialization and de-initialization functions .....	499
31.2.7 Input and Output functions .....	500
31.2.8 Control functions .....	504
31.2.9 State functions .....	505
31.3 NOR Firmware driver defines .....	506
31.3.1 NOR .....	506
<b>32 HAL PCCARD Generic Driver .....</b>	<b>508</b>
32.1 PCCARD Firmware driver registers structures .....	508
32.1.1 PCCARD_HandleTypeDef .....	508
32.2 PCCARD Firmware driver API description .....	508
32.2.1 How to use this driver .....	508
32.2.2 PCCARD Initialization and de-initialization functions .....	509

32.2.3	PCCARD Input and Output functions .....	509
32.2.4	PCCARD State functions.....	509
32.2.5	Initialization and de-initialization functions .....	509
32.2.6	Input and Output functions .....	511
32.2.7	State functions.....	514
32.3	PCCARD Firmware driver defines.....	515
32.3.1	PCCARD .....	515
<b>33</b>	<b>HAL PCD Generic Driver .....</b>	<b>518</b>
33.1	PCD Firmware driver registers structures .....	518
33.1.1	PCD_HandleTypeDef .....	518
33.2	PCD Firmware driver API description.....	518
33.2.1	How to use this driver .....	518
33.2.2	Initialization and de-initialization functions .....	519
33.2.3	IO operation functions .....	519
33.2.4	Peripheral Control functions .....	519
33.2.5	Peripheral State functions .....	520
33.2.6	Initialization and de-initialization functions .....	520
33.2.7	IO operation functions .....	521
33.2.8	Peripheral Control functions .....	526
33.2.9	Peripheral State functions .....	531
33.3	PCD Firmware driver defines .....	532
33.3.1	PCD .....	532
<b>34</b>	<b>HAL PWR Generic Driver .....</b>	<b>535</b>
34.1	PWR Firmware driver registers structures .....	535
34.1.1	PWR_PVDTTypeDef .....	535
34.1.2	PWR_TypeDef.....	535
34.2	PWR Firmware driver API description.....	535
34.2.1	Initialization and de-initialization functions .....	535
34.2.2	Peripheral Control functions .....	536
34.2.3	Initialization and de-initialization functions .....	538
34.2.4	Peripheral Control functions .....	539
34.3	PWR Firmware driver defines .....	543
34.3.1	PWR .....	543
<b>35</b>	<b>HAL PWR Extension Driver .....</b>	<b>546</b>
35.1	PWREx Firmware driver API description.....	546
35.1.1	Peripheral extended features functions .....	546

Contents	UM1725
35.1.2 Peripheral Extended features functions .....	547
35.2 PWREx Firmware driver defines .....	549
35.2.1 PWREx .....	549
<b>36 HAL RCC Generic Driver.....</b>	<b>551</b>
36.1 RCC Firmware driver registers structures .....	551
36.1.1 RCC_PLLInitTypeDef .....	551
36.1.2 RCC_ClkInitTypeDef .....	551
36.1.3 RCC_OsclInitTypeDef .....	552
36.2 RCC Firmware driver API description .....	553
36.2.1 RCC specific features .....	553
36.2.2 Initialization and de-initialization functions .....	553
36.2.3 Peripheral Control functions .....	554
36.2.4 Initialization and de-initialization functions .....	555
36.2.5 Peripheral Control functions .....	556
36.3 RCC Firmware driver defines .....	561
36.3.1 RCC .....	561
<b>37 HAL RCC Extension Driver .....</b>	<b>575</b>
37.1 RCCEEx Firmware driver registers structures .....	575
37.1.1 RCC_PLLI2SInitTypeDef .....	575
37.1.2 RCC_PLLSAllInitTypeDef .....	575
37.1.3 RCC_PeriphCLKInitTypeDef .....	576
37.2 RCCEEx Firmware driver API description .....	577
37.2.1 Extended Peripheral Control functions .....	577
37.2.2 Extended Peripheral Control functions .....	577
37.3 RCCEEx Firmware driver defines .....	578
37.3.1 RCCEEx .....	578
<b>38 HAL RNG Generic Driver.....</b>	<b>581</b>
38.1 RNG Firmware driver registers structures .....	581
38.1.1 RNG_HandleTypeDef .....	581
38.1.2 RNG_TypeDef .....	581
38.2 RNG Firmware driver API description .....	582
38.2.1 How to use this driver .....	582
38.2.2 Initialization and de-initialization functions .....	582
38.2.3 Peripheral Control functions .....	582
38.2.4 Peripheral State functions .....	582
38.2.5 Initialization and de-initialization functions .....	582
38.2.6 Peripheral Control functions .....	584

38.2.7	Peripheral State functions .....	586
38.3	RNG Firmware driver defines.....	586
38.3.1	RNG.....	586
<b>39</b>	<b>HAL RTC Generic Driver .....</b>	<b>588</b>
39.1	RTC Firmware driver registers structures .....	588
39.1.1	RTC_HandleTypeDef .....	588
39.1.2	RTC_InitTypeDef.....	588
39.1.3	RTC_DateTypeDef.....	589
39.1.4	RTC_TimeTypeDef.....	589
39.1.5	RTC_AlarmTypeDef .....	590
39.1.6	RTC_TypeDef.....	591
39.2	RTC Firmware driver API description.....	593
39.2.1	Backup Domain Operating Condition.....	594
39.2.2	Backup Domain Reset.....	594
39.2.3	Backup Domain Access.....	594
39.2.4	How to use this driver .....	595
39.2.5	RTC and low power modes .....	595
39.2.6	Initialization and de-initialization functions .....	595
39.2.7	RTC Time and Date functions .....	596
39.2.8	RTC Alarm functions .....	596
39.2.9	Peripheral Control functions .....	596
39.2.10	Peripheral State functions .....	596
39.2.11	Initialization and de-initialization functions .....	597
39.2.12	RTC Time and Date functions .....	598
39.2.13	RTC Alarm functions .....	600
39.2.14	Peripheral Control functions .....	603
39.2.15	Peripheral State functions .....	604
39.3	RTC Firmware driver defines .....	605
39.3.1	RTC .....	605
<b>40</b>	<b>HAL RTC Extension Driver .....</b>	<b>614</b>
40.1	RTCEEx Firmware driver registers structures .....	614
40.1.1	RTC_TamperTypeDef .....	614
40.2	RTCEEx Firmware driver API description.....	615
40.2.1	How to use this driver .....	615
40.2.2	RTC TimeStamp and Tamper functions.....	616
40.2.3	RTC Wake-up functions .....	616
40.2.4	Extension Peripheral Control functions .....	616

Contents	UM1725
40.2.5 Extended features functions .....	617
40.2.6 RTC TimeStamp and Tamper functions .....	617
40.2.7 RTC Wake-up functions .....	623
40.2.8 Extension Peripheral Control functions .....	625
40.2.9 Extended features functions .....	631
40.3 RTCEx Firmware driver defines .....	631
40.3.1 RTCEx .....	631
<b>41 HAL SAI Generic Driver .....</b>	<b>639</b>
41.1 SAI Firmware driver registers structures .....	639
41.1.1 SAI_HandleTypeDef.....	639
41.1.2 SAI_InitTypeDef .....	640
41.1.3 SAI_FrameInitTypeDef .....	641
41.1.4 SAI_SlotInitTypeDef .....	642
41.1.5 SAI_Block_TypeDef .....	642
41.1.6 SAI_TypeDef .....	643
41.2 SAI Firmware driver API description .....	643
41.2.1 How to use this driver .....	643
41.2.2 Initialization and de-initialization functions .....	645
41.2.3 IO operation functions .....	646
41.2.4 Peripheral State and Errors functions .....	646
41.2.5 Initialization and de-initialization functions .....	647
41.2.6 IO operation functions .....	648
41.2.7 Peripheral State functions .....	654
41.3 SAI Firmware driver defines .....	654
41.3.1 SAI .....	654
<b>42 HAL SMARTCARD Generic Driver.....</b>	<b>664</b>
42.1 SMARTCARD Firmware driver registers structures .....	664
42.1.1 SMARTCARD_HandleTypeDef.....	664
42.1.2 SMARTCARD_InitTypeDef .....	664
42.1.3 USART_TypeDef.....	666
42.2 SMARTCARD Firmware driver API description.....	666
42.2.1 How to use this driver .....	666
42.2.2 Initialization and Configuration functions.....	668
42.2.3 IO operation functions .....	669
42.2.4 Peripheral State and Errors functions .....	670
42.2.5 SmartCard Initialization and de-initialization functions.....	670
42.2.6 IO operation functions .....	672

42.2.7	Peripheral State and Errors functions .....	676
42.3	SMARTCARD Firmware driver defines .....	676
42.3.1	SMARTCARD.....	676
<b>43</b>	<b>HAL SRAM Generic Driver.....</b>	<b>681</b>
43.1	SRAM Firmware driver registers structures.....	681
43.1.1	SRAM_HandleTypeDef .....	681
43.2	SRAM Firmware driver API description.....	681
43.2.1	How to use this driver .....	681
43.2.2	SRAM Initialization and de_initialization functions .....	682
43.2.3	SRAM Input and Output functions.....	682
43.2.4	SRAM Control functions .....	683
43.2.5	SRAM State functions .....	683
43.2.6	Initialization and de-initialization functions .....	683
43.2.7	Input and Output functions .....	685
43.2.8	Control functions.....	688
43.2.9	State functions.....	689
43.3	SRAM Firmware driver defines .....	690
43.3.1	SRAM .....	690
<b>44</b>	<b>HAL SDRAM Generic Driver .....</b>	<b>691</b>
44.1	SDRAM Firmware driver registers structures .....	691
44.1.1	SDRAM_HandleTypeDef.....	691
44.2	SDRAM Firmware driver API description .....	691
44.2.1	How to use this driver .....	691
44.2.2	SDRAM Initialization and de_initialization functions .....	692
44.2.3	SDRAM Input and Output functions .....	692
44.2.4	SDRAM Control functions.....	692
44.2.5	SDRAM State functions .....	693
44.2.6	Initialization and de-initialization functions .....	693
44.2.7	Input and Output functions .....	696
44.2.8	Control functions.....	699
44.2.9	State functions.....	702
44.3	SDRAM Firmware driver defines.....	702
44.3.1	SDRAM.....	702
<b>45</b>	<b>HAL SPI Generic Driver.....</b>	<b>703</b>
45.1	SPI Firmware driver registers structures .....	703
45.1.1	SPI_HandleTypeDef.....	703

45.1.2	SPI_InitTypeDef .....	703
45.1.3	SPI_TypeDef .....	705
45.2	SPI Firmware driver API description .....	705
45.2.1	How to use this driver .....	705
45.2.2	Initialization and de-initialization functions .....	706
45.2.3	IO operation functions .....	706
45.2.4	Peripheral State and Errors functions .....	707
45.2.5	Initialization and de-initialization functions .....	708
45.2.6	IO operation functions .....	709
45.2.7	Peripheral State and Errors functions .....	714
45.3	SPI Firmware driver defines .....	715
45.3.1	SPI .....	715
<b>46</b>	<b>HAL TIM Generic Driver .....</b>	<b>720</b>
46.1	TIM Firmware driver registers structures .....	720
46.1.1	TIM_HandleTypeDef .....	720
46.1.2	TIM_Base_InitTypeDef .....	720
46.1.3	TIM_OC_InitTypeDef .....	721
46.1.4	TIM_IC_InitTypeDef .....	722
46.1.5	TIM_OnePulse_InitTypeDef .....	722
46.1.6	TIM_ClockConfigTypeDef .....	723
46.1.7	TIM_ClearInputConfigTypeDef .....	724
46.1.8	TIM_SlaveConfigTypeDef .....	724
46.1.9	TIM_Encoder_InitTypeDef .....	725
46.1.10	TIM_TypeDef .....	726
46.2	TIM Firmware driver API description .....	727
46.2.1	TIMER Generic features .....	727
46.2.2	How to use this driver .....	728
46.2.3	Time Base functions .....	729
46.2.4	Peripheral State functions .....	729
46.2.5	Time Output Compare functions .....	729
46.2.6	Time PWM functions .....	730
46.2.7	Time Input Capture functions .....	730
46.2.8	Time One Pulse functions .....	731
46.2.9	Time Encoder functions .....	731
46.2.10	IRQ handler management .....	731
46.2.11	Peripheral Control functions .....	732
46.2.12	TIM Callbacks functions .....	732
46.2.13	Time Base functions .....	732

46.2.14	Peripheral State functions .....	736
46.2.15	Time Output Compare functions .....	738
46.2.16	Time PWM functions .....	742
46.2.17	Time Input Capture functions .....	747
46.2.18	Time One Pulse functions .....	751
46.2.19	Time Encoder functions.....	754
46.2.20	TIM IRQ handler management.....	758
46.2.21	Peripheral Control functions .....	759
46.2.22	TIM Callbacks functions .....	767
46.3	TIM Firmware driver defines.....	769
46.3.1	TIM.....	769
<b>47</b>	<b>HAL TIM Extension Driver.....</b>	<b>786</b>
47.1	TIMEx Firmware driver registers structures.....	786
47.1.1	TIM_MasterConfigTypeDef .....	786
47.1.2	TIM_HallSensor_InitTypeDef .....	786
47.1.3	TIM_BreakDeadTimeConfigTypeDef .....	787
47.2	TIMEx Firmware driver API description .....	787
47.2.1	TIMER Extended features .....	787
47.2.2	How to use this driver .....	788
47.2.3	Timer Hall Sensor functions .....	788
47.2.4	Timer Complementary Output Compare functions.....	789
47.2.5	Timer Complementary PWM functions.....	789
47.2.6	Timer Complementary One Pulse functions.....	790
47.2.7	Peripheral Control functions .....	790
47.2.8	Extension Callbacks functions.....	790
47.2.9	Extension Peripheral State functions .....	790
47.2.10	Timer Hall Sensor functions .....	791
47.2.11	Timer Complementary Output Compare functions.....	794
47.2.12	Timer Complementary PWM functions.....	797
47.2.13	Timer Complementary One Pulse functions.....	800
47.2.14	Peripheral Control functions .....	801
47.2.15	Extension Callbacks functions.....	805
47.2.16	Extension Peripheral State functions .....	806
47.3	TIMEx Firmware driver defines .....	806
47.3.1	TIMEx .....	806
<b>48</b>	<b>HAL UART Generic Driver.....</b>	<b>811</b>
48.1	UART Firmware driver registers structures .....	811

Contents	UM1725
48.1.1    UART_HandleTypeDef .....	811
48.1.2    UART_InitTypeDef .....	811
48.1.3    USART_TypeDef.....	812
48.2    UART Firmware driver API description .....	813
48.2.1    How to use this driver .....	813
48.2.2    Initialization and Configuration functions.....	815
48.2.3    IO operation functions .....	816
48.2.4    Peripheral Control functions .....	817
48.2.5    Peripheral State and Errors functions .....	817
48.2.6    Initialization and de-initialization functions .....	817
48.2.7    IO operation functions .....	820
48.2.8    Peripheral Control functions .....	826
48.2.9    Peripheral State and Errors functions .....	828
48.3    UART Firmware driver defines .....	829
48.3.1    USART .....	829
<b>49 HAL USART Generic Driver .....</b>	<b>833</b>
49.1    USART Firmware driver registers structures.....	833
49.1.1    USART_HandleTypeDef .....	833
49.1.2    USART_InitTypeDef .....	833
49.1.3    USART_TypeDef.....	834
49.2    USART Firmware driver API description .....	835
49.2.1    How to use this driver .....	835
49.2.2    Initialization and Configuration functions.....	837
49.2.3    IO operation functions .....	837
49.2.4    Peripheral State and Errors functions .....	839
49.2.5    USART Initialization and de-initialization functions .....	839
49.2.6    IO operation functions .....	840
49.2.7    Peripheral State and Errors functions .....	848
49.3    USART Firmware driver defines.....	849
49.3.1    USART.....	849
<b>50 HAL WWDG Generic Driver .....</b>	<b>853</b>
50.1    WWDG Firmware driver registers structures.....	853
50.1.1    WWDG_HandleTypeDef .....	853
50.1.2    WWDG_InitTypeDef .....	853
50.1.3    WWDG_TypeDef.....	854
50.2    WWDG Firmware driver API description .....	854
50.2.1    Initialization and de-initialization functions .....	854

50.2.2	IO operation functions .....	854
50.2.3	Peripheral State functions .....	855
50.2.4	Initialization and de-initialization functions .....	855
50.2.5	IO operation functions .....	856
50.2.6	Peripheral State functions .....	858
50.3	WWDG Firmware driver defines.....	859
50.3.1	WWDG.....	859
<b>51</b>	<b>FAQs.....</b>	<b>860</b>
<b>52</b>	<b>Revision history .....</b>	<b>864</b>

## List of tables

Table 1: Acronyms and definitions .....	26
Table 2: HAL drivers files .....	28
Table 3: User-application files .....	30
Table 4: APis classification .....	34
Table 5: List of devices supported by HAL drivers .....	34
Table 6: HAL API naming rules .....	37
Table 7: Macros handling interrupts and specific clock configurations .....	38
Table 8: Callback functions .....	39
Table 9: HAL generic APIs .....	40
Table 10: HAL extension APIs .....	41
Table 11: Define statements used for HAL configuration .....	45
Table 12: Description of GPIO_InitTypeDef structure .....	47
Table 13: Description of EXTI configuration macros .....	50
Table 14: MSP functions .....	54
Table 15: Timeout values .....	58
Table 16: Document revision history .....	864

## List of figures

Figure 1: Example of project template .....	31
Figure 2: Adding device-specific functions .....	42
Figure 3: Adding family-specific functions .....	42
Figure 4: Adding new peripherals .....	43
Figure 5: Updating existing APIs .....	43
Figure 6: File inclusion model .....	44
Figure 7: HAL driver model .....	52

# 1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
CAN	Controller area network
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRYP	Cryptographic processor unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DCMI	Digital Camera Module Interface
DMA	Direct Memory Access
DMA2D	Chrom-Art Accelerator™ controller
ETH	Ethernet controller
EXTI	External interrupt/event controller
FLASH	Flash memory
FSMC	Flexible Static Memory controller
FMC	Flexible Memory controller
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB Host Controller Driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LTDC	LCD TFT Display Controller
MSP	MCU Specific Package
NAND	NAND external Flash memory
NOR	NOR external Flash memory
NVIC	Nested Vectored Interrupt Controller
PCCARD	PCCARD external memory
PCD	USB Peripheral Controller Driver

Acronym	Definition
PWR	Power controller
RCC	Reset and clock controller
RNG	Random Number Generator
RTC	Real-time clock
SAI	Serial Audio Interface
SD	Secure Digital
SDRAM	SDRAM external memory
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

## 2 Overview of HAL drivers

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

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

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

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode.
- 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
<i>stm32f4xx_hal_ppp.c</i>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32f4xx_hal_adc.c, stm32f4xx_hal_irda.c, ...</i>
<i>stm32f4xx_hal_ppp.h</i>	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. <i>Example: stm32f4xx_hal_adc.h, stm32f4xx_hal_irda.h, ...</i>

File	Description
<i>stm32f4xx_hal_ppp_ex.c</i>	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. <i>Example: stm32f4xx_hal_adc_ex.c, stm32f4xx_hal_dma_ex.c, ...</i>
<i>stm32f4xx_hal_ppp_ex.h</i>	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs <i>Example: stm32f4xx_hal_adc_ex.h, stm32f4xx_hal_dma_ex.h, ...</i>
<i>stm32f4xx_ll_ppp.c</i>	Peripheral low layer driver that can be accessed from one or more HAL drivers. It offers a set of APIs and services used by the upper driver. From the user point of view, low-level drivers are not accessible directly. They are used only by the HAL drivers built upon them. <i>Example: stm32f4xx_ll_fmc.c offers a set of API used by stm32f4xx_hal_sdram.c, stm32f4xx_hal_sram.c, stm32f4xx_hal_nor.c, stm32f4xx_hal_nand.c, ...</i>
<i>stm32f4xx_ll_ppp.h</i>	Header file of the low layer C file. It is included in the HAL driver header file, thus making the low-level driver an intrinsic add-on of the HAL driver that is not visible from the application. <i>Example: stm32f4xx_ll_fmc.h, stm32f4xx_ll_usb.h, ...</i>
<i>stm32f4xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32f4xx_hal.h</i>	stm32f4xx_hal.c header file
<i>stm32f4xx_hal_msp_template.c</i>	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.
<i>stm32f4xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32f4xx_hal_def.h</i>	Common HAL resources such as common define statements, enumerations, structures and macros.



Since the low level drivers are only used by the HAL drivers built upon them, the APIs provided by these drivers will not be described in this document.

## 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
<i>system_stm32f4xx.c</i>	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 : <ul style="list-style-type: none"> <li>• relocate the vector table in internal SRAM.</li> <li>• configure FSMC/FMC peripheral (when available) to use as data memory the external SRAM or SDRAM mounted on the evaluation board.</li> </ul>
<i>startup_stm32f4xx.s</i>	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.
<i>stm32f4xx_flash.icf (optional)</i>	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32f4xx_hal_msp.c</i>	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32f4xx_hal_conf.h</i>	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.
<i>stm32f4xx_it.c/h</i>	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 <i>stm32f4xx_hal.c</i> ) 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.
<i>main.c/h</i>	This file contains the main program routine, mainly: <ul style="list-style-type: none"> <li>• the call to HAL_Init()</li> <li>• assert_failed() implementation</li> <li>• system clock configuration</li> <li>• peripheral HAL initialization and user application code.</li> </ul>

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.

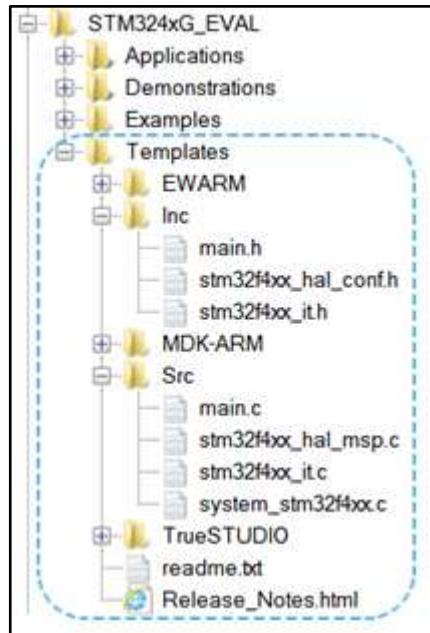
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- 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 simultaneous.

***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 a subroutine 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 whether the Receive or Transmit mode is enabled or
disabled.*/
uint32_t HwFlowCtl; /*!< Specifies whether the hardware flow control mode is enabled
or disabled.*/
uint32_t OverSampling; /*!< Specifies whether the Over sampling 8 is enabled or
disabled,
to achieve higher speed (up to 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_InjectedStop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT(ADC_HandleTypeDef* hadc);

```

- **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.

```

#ifndef STM32F427xx
#define STM32F427xx
#ifndef STM32F437xx
#define STM32F437xx
#ifndef STM32F429xx
#define STM32F429xx
#ifndef STM32F439xx
#define STM32F439xx
HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP(void);
HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP(void);
#endif /* STM32F427xx || STM32F437xx || STM32F429xx || STM32F439xx */
#endif /* STM32F439xx */
#endif /* STM32F429xx */
#endif /* STM32F437xx */
#endif /* STM32F427xx */

```

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 <sup>(1)</sup>
Family specific APIs		X
Device specific APIs		X

**Notes:**

<sup>(1)</sup>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	STM32F 405xx	STM32F 415xx	STM32F 407xx	STM32F 417xx	STM32F 427xx	STM32F 437xx	STM32F 429xx	STM32F 439xx	STM32F 401xC	STM32F 401xE
stm32f4xx_hal.c	Yes									
stm32f4xx_hal_adc.c	Yes									
stm32f4xx_hal_adc_ex.c	Yes									
stm32f4xx_hal_can.c	Yes	No	No							
stm32f4xx_hal_cortex.c	Yes									
stm32f4xx_hal_crc.c	Yes									
stm32f4xx_hal_cryp.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No
stm32f4xx_hal_cryp_ex.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No
stm32f4xx_hal_dac.c	Yes	No	No							
stm32f4xx_hal_dac_ex.c	Yes	No	No							

IP/Module	STM32F 405xx	STM32F 415xx	STM32F 407xx	STM32F 417xx	STM32F 427xx	STM32F 437xx	STM32F 429xx	STM32F 439xx	STM32F 401xC	STM32F 401xE
stm32f4xx_hal_dcmi.c	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_dma.c	Yes									
stm32f4xx_hal_dma2d.c	No	No	No	No	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_dma_ex.c	Yes									
stm32f4xx_hal_eth.c	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No
stm32f4xx_hal_flash.c	Yes									
stm32f4xx_hal_flash_ex.c	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_gpio.c	Yes									
stm32f4xx_hal_hash.c	No	Yes	No	Yes	No	Yes	No	Yes	No	No
stm32f4xx_hal_hash_ex.c	No	No	No	No	No	Yes	No	Yes	No	No
stm32f4xx_hal_hcd.c	Yes									
stm32f4xx_hal_i2c.c	Yes									
stm32f4xx_hal_i2c_ex.c	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32f4xx_hal_i2s.c	Yes									
stm32f4xx_hal_i2s_rda.c	Yes									
stm32f4xx_hal_iwdg.c	Yes									
stm32f4xx_hal_ltdc.c	No	No	No	No	No	No	Yes	Yes	No	No
stm32f4xx_hal_nand.c	Yes	No	No							
stm32f4xx_hal_nor.c	Yes	No	No							
stm32f4xx_hal_pccard.c	Yes	No	No							
stm32f4xx_hal_pcd.c	Yes									
stm32f4xx_hal_pwr.c	Yes									

## Overview of HAL drivers

**UM1725**

IP/Module	STM32F 405xx	STM32F 415xx	STM32F 407xx	STM32F 417xx	STM32F 427xx	STM32F 437xx	STM32F 429xx	STM32F 439xx	STM32F 401xC	STM32F 401xE
<code>stm32f4xx_hal_pwr_ex.c</code>	No	No	No	No	Yes	Yes	Yes	Yes	No	No
<code>stm32f4xx_hal_rcc.c</code>	Yes									
<code>stm32f4xx_hal_rcc_ex.c</code>	Yes									
<code>stm32f4xx_hal_rng.c</code>	Yes									
<code>stm32f4xx_hal_rtc.c</code>	Yes									
<code>stm32f4xx_hal_rtc_ex.c</code>	Yes									
<code>stm32f4xx_hal_sai.c</code>	No	No	No	No	Yes	Yes	Yes	Yes	No	No
<code>stm32f4xx_hal_sd.c</code>	Yes									
<code>stm32f4xx_hal_sdram.c</code>	No	No	No	No	Yes	Yes	Yes	Yes	No	No
<code>stm32f4xx_hal_smartcard.c</code>	Yes									
<code>stm32f4xx_hal_spi.c</code>	Yes									
<code>stm32f4xx_hal_sram.c</code>	Yes	No	No							
<code>stm32f4xx_hal_tim.c</code>	Yes									
<code>stm32f4xx_hal_tim_ex.c</code>	Yes									
<code>stm32f4xx_hal_uart.c</code>	Yes									
<code>stm32f4xx_hal_usart.c</code>	Yes									
<code>stm32f4xx_hal_wwdg.c</code>	Yes									
<code>stm32f4xx_ll_fmc.c</code>	No	No	No	No	Yes	Yes	Yes	Yes	No	No
<code>stm32f4xx_ll_fs_mc.c</code>	Yes	Yes	Yes	Yes	No	No	No	No	No	No
<code>stm32f4xx_ll_sdmmc.c</code>	Yes									
<code>stm32f4xx_ll_usb.c</code>	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	<i>stm32f4xx_hal_ppp (c/h)</i>	<i>stm32f4xx_hal_ppp_ex (c/h)</i>	<i>stm32f4xx_hal_ppp_ex (c/h)</i>
Module name	<i>HAL_PPP_MODULE</i>		
Function name	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEX_Function</i> <i>HAL_PPPEX_FeatureFunction_MODE</i>	<i>HAL_PPPEX_Function</i> <i>HAL_PPPEX_FeatureFunction_MODE</i>
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	NA	NA

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with \_TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32F4xx reference manuals.
- Peripheral registers are declared in the PPP\_TypeDef structure (e.g. ADC\_TypeDef) in stm32f4xxx.h header file. stm32f4xxx.h corresponds to stm32f401xc.h, stm32f401xe.h, stm32f405xx.h, stm32f415xx.h, stm32f407xx.h, stm32f417xx.h, stm32f427xx.h, stm32f437xx.h, stm32f429xx.h or stm32f439xx.h
- Peripheral function names are prefixed by HAL\_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. HAL\_UART\_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP\_InitTypeDef (e.g. ADC\_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP\_xxxxConfTypeDef (e.g. ADC\_ChannelConfTypeDef).
- Peripheral handle structures are named PPP\_HandleTypeDef (e.g DMA\_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP\_InitTypeDef are named HAL\_PPP\_Init (e.g. HAL\_TIM\_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named PPP\_DeInit, e.g. TIM\_DeInit.
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL\_PPP\_Function\_DMA ()*.

- The **Feature** prefix should refer to the new feature.  
Example: *HAL\_ADC\_InjectionStart()* 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
<code>_HAL_PPP_ENABLE_IT(_HANDLE_, _INTERRUPT_)</code>	Enables a specific peripheral interrupt
<code>_HAL_PPP_DISABLE_IT(_HANDLE_, _INTERRUPT_)</code>	Disables a specific peripheral interrupt
<code>_HAL_PPP_GET_IT (_HANDLE_, _INTERRUPT_)</code>	Gets a specific peripheral interrupt status
<code>_HAL_PPP_CLEAR_IT (_HANDLE_, _INTERRUPT_)</code>	Clears a specific peripheral interrupt status
<code>_HAL_PPP_GET_FLAG (_HANDLE_, _FLAG_)</code>	Gets a specific peripheral flag status
<code>_HAL_PPP_CLEAR_FLAG (_HANDLE_, _FLAG_)</code>	Clears a specific peripheral flag status
<code>_HAL_PPP_ENABLE(_HANDLE_)</code>	Enables a peripheral
<code>_HAL_PPP_DISABLE(_HANDLE_)</code>	Disables a peripheral
<code>_HAL_PPP_XXXX (_HANDLE_, _PARAM_)</code>	Specific PPP HAL driver macro
<code>_HAL_PPP_GET_IT_SOURCE (_HANDLE_, _INTERRUPT_)</code>	Checks the source of specified interrupt

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the `stm32f4xx_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 = ".

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

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

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

```
#define HAL_LINKDMA( HANDLE , PPP DMA FIELD , DMA HANDLE ) \
do{ \
    ( HANDLE )-> PPP DMA FIELD = &( DMA HANDLE ); \
    ( DMA HANDLE ).Parent = ( HANDLE ); \
} while(0)
```

### 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 stm32f4xx\_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\_MspDelt()
- Process complete callbacks : HAL\_PPP\_ProcessCpltCallback
- Error callback: HAL\_PPP\_ErrorCallback.

**Table 8: Callback functions**

Callback functions	Example
HAL_PPP_MspInit() / _Delt()	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\_Delt()
- 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\_DelInit()* function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

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

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

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

**Table 9: HAL generic APIs**

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

Function Group	Common API Name	Description
	<code>HAL_ADC_AnalogWDGConfig</code>	This function configures the analog watchdog for the selected ADC
<i>State and Errors group</i>	<code>HAL_ADC_GetState()</code>	This function allows getting in runtime the peripheral and the data flow states.
	<code>HAL_ADC_GetError()</code>	This function 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, `stm32f4xx_hal_ppp_ex.c`, that includes all the specific functions and define statements (`stm32f4xx_hal_ppp_ex.h`) for a given part number.

Below an example based on the ADC peripheral:

Table 10: HAL extension APIs

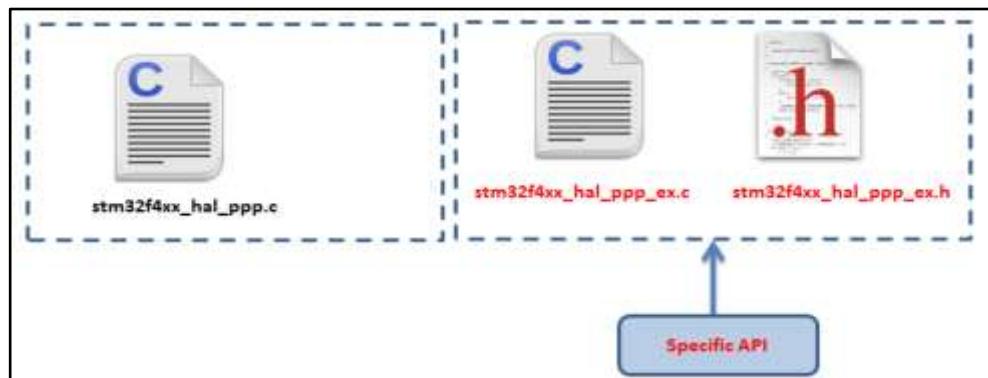
Function Group	Common API Name
<code>HAL_ADCEx_InjectedStart()</code>	This function starts injected channel ADC conversions when the polling method is used
<code>HAL_ADCEx_InjectedStop()</code>	This function stops injected channel ADC conversions when the polling method is used
<code>HAL_ADCEx_InjectedStart_IT()</code>	This function starts injected channel ADC conversions when the interrupt method is used
<code>HAL_ADCEx_InjectedStop_IT()</code>	This function stops injected channel ADC conversions when the interrupt method is used
<code>HAL_ADCEx_InjectedConfigChannel()</code>	This function configures the selected ADC Injected channel (corresponding rank in the sequencer and sample time)

### 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 `stm32f4xx_hal_ppp_ex.c` extension file. They are named `HAL_PPPEX_Function()`.

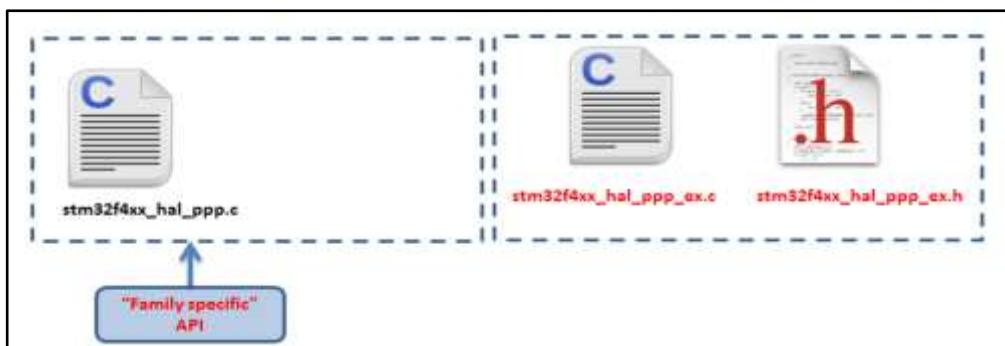
**Figure 2: Adding device-specific functions**

Example: `stm32f4xx_hal_flash_ex.c/h`

```
#if defined(STM32F427xx) || defined(STM32F437xx) || defined(STM32F429xx) ||  
defined(STM32F439xx)  
HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP(void);  
HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP(void);  
#endif /* STM32F427xx || STM32F437xx || STM32F429xx || STM32F439xx || */
```

### 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**

Example: `stm32f4xx_hal_adc_ex.c/h`

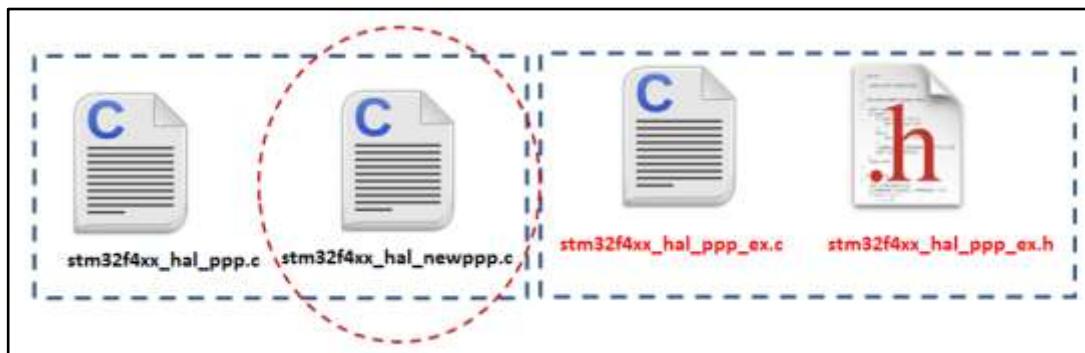
```
HAL_StatusTypeDef HAL_ADCEx_InjectedStop(ADC_HandleTypeDef* hadc);  
HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT(ADC_HandleTypeDef* hadc);  
HAL_StatusTypeDef HAL_ADCEx_InjectedStart(ADC_HandleTypeDef* hadc);  
HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT(ADC_HandleTypeDef* hadc);
```

### 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 `stm32f4xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32fxx_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

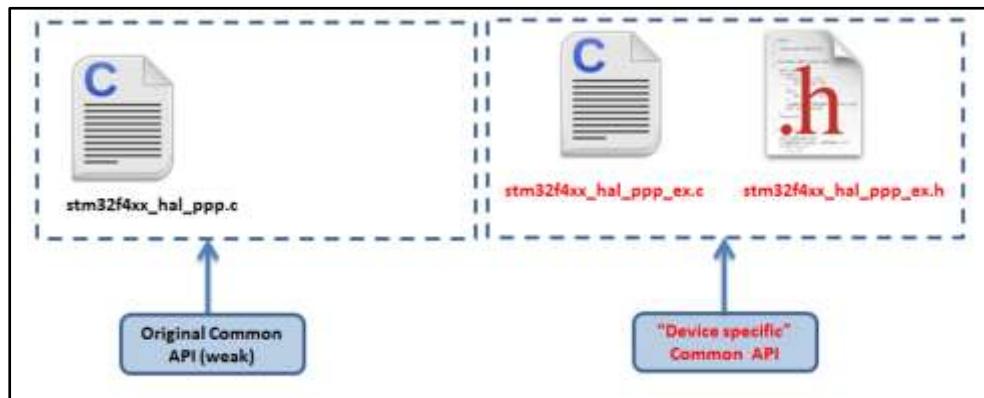


Example: `stm32f4xx_hal_sai.c/h`

#### Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the `stm32f4xx_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.

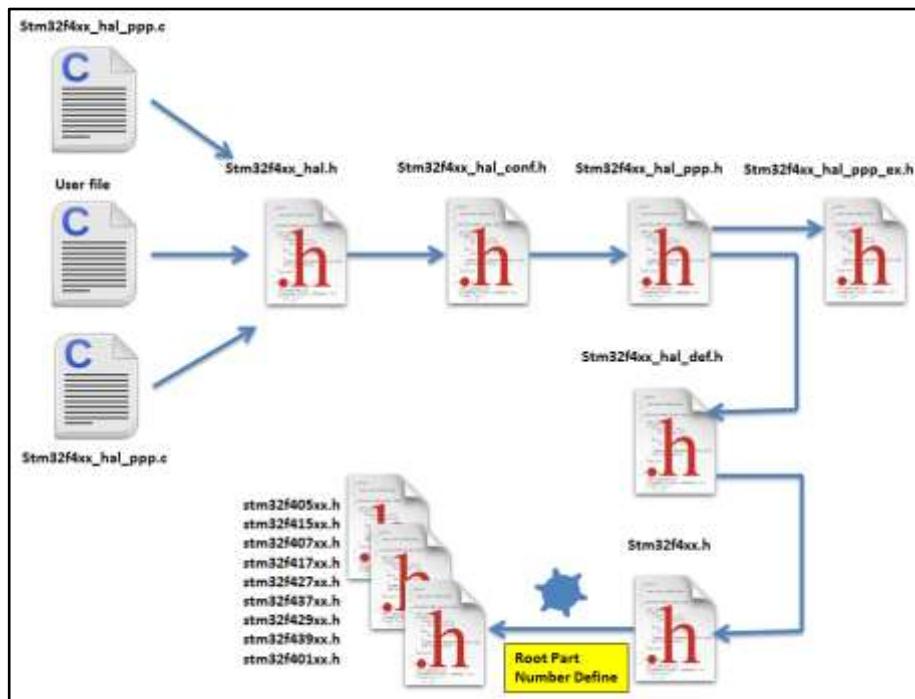
Example:

```
#if defined (STM32F401xx)
typedef struct
{
(...)
}PPP_InitTypeDef;
#endif /* STM32F401xx */
```

## 2.8 File inclusion model

The header of the common HAL driver file (`stm32f4xx_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 stm32f4xx_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 *stm32f4xx\_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 `stm32f4xx_hal_def.h` file calls the `stm32f4xx.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, `stm32f4xx_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
<b>HSE_VALUE</b>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 (Hz)
<b>HSE_STARTUP_TIMEOUT</b>	Timeout for HSE start up, expressed in ms	5000
<b>HSI_VALUE</b>	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
<b>EXTERNAL_CLOCK_VALUE</b>	This value is used by the I2S/SAI HAL module to compute the I2S/SAI clock source frequency, this source is inserted directly through I2S_CKIN pad.	12288000 (Hz)
<b>VDD_VALUE</b>	VDD value	3300 (mV)

Configuration item	Description	Default Value
<b>USERTOS</b>	Enables the use of RTOS	FALSE (for future use)
<b>PREFETCH_ENABLE</b>	Enables prefetch feature	TRUE
<b>INSTRUCTION_CACHE_ENABLE</b>	Enables instruction cache	TRUE
<b>DATA_CACHE_ENABLE</b>	Enables data cache	TRUE
<b>USE HAL PPP MODULE</b>	Enables module to be used in the HAL driver	
<b>MAC_ADDRx</b>	Ethernet peripheral configuration : MAC address	
<b>ETH_RX_BUF_SIZE</b>	Ethernet buffer size for receive	ETH_MAX_PACKET_SIZE
<b>ETH_TX_BUF_SIZE</b>	Ethernet buffer size for transmit	ETH_MAX_PACKET_SIZE
<b>ETH_RXBUFN</b>	The number of Rx buffers of size ETH_RX_BUF_SIZE	4
<b>ETH_TXBUFN</b>	The number of Tx buffers of size ETH_RX_BUF_SIZE	4
<b>DP83848_PHY_ADDRESS</b>	DB83848 Ethernet PHY Address	0x01
<b>PHY_RESET_DELAY</b>	PHY Reset delay these values are based on a 1 ms Systick interrupt	0x000000FF
<b>PHY_CONFIG_DELAY</b>	PHY Configuration delay	0x00000FFF
<b>PHY_BCR PHY_BSR</b>	Common PHY Registers	
<b>PHY_SR PHY_MICR PHY_MISR</b>	Extended PHY registers	



The `stm32f4xx_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 `stm32f4xx_hal_conf_template.h` file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

## 2.11 HAL system peripheral handling

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

### 2.11.1 Clock

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

- `HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct)`. This function configures/enables multiple clock sources (HSE, HSI, LSE, LSI, PLL).

- HAL\_RCC\_ClockConfig (RCC\_ClkInitTypeDef \*RCC\_ClkInitStruct, uint32\_t FLatency). This function
  - Selects the system clock source
  - Configures AHB, APB1 and APB2 clock dividers
  - Configures the number of Flash memory wait states
  - Updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (RTC, SDIO, I2S, SAI, Audio PLL...). In this case, the clock configuration is performed by an extended API defined in `stm32f4xx_hal_ppp_ex.c`: `HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)`.

Additional RCC HAL driver functions are available:

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

A set of macros are defined in `stm32f4xx_hal_rcc.h`. They allows 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\_DelInit()
- 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 `stm32f4xx_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: <code>GPIO_PIN_x</code> or <code>GPIO_PIN_All</code> , where <code>x[0..15]</code>

Structure field	Description
Mode	<p>Specifies the operating mode for the selected pins: GPIO mode or EXTI mode.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>• <u>GPIO mode</u> <ul style="list-style-type: none"> <li>– GPIO_MODE_INPUT : Input Floating</li> <li>– GPIO_MODE_OUTPUT_PP : Output Push Pull</li> <li>– GPIO_MODE_OUTPUT_OD : Output Open Drain</li> <li>– GPIO_MODE_AF_PP : Alternate Function Push Pull</li> <li>– GPIO_MODE_AF_OD : Alternate Function Open Drain</li> <li>– GPIO_MODE_ANALOG : Analog mode</li> </ul> </li> <li>• <u>External Interrupt Mode</u> <ul style="list-style-type: none"> <li>– GPIO_MODE_IT_RISING : Rising edge trigger detection</li> <li>– GPIO_MODE_IT_FALLING : Falling edge trigger detection</li> <li>– GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection</li> </ul> </li> <li>• <u>External Event Mode</u> <ul style="list-style-type: none"> <li>– GPIO_MODE_EVT_RISING : Rising edge trigger detection</li> <li>– GPIO_MODE_EVT_FALLING : Falling edge trigger detection</li> <li>– GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection</li> </ul> </li> </ul>
Pull	<p>Specifies the Pull-up or Pull-down activation for the selected pins.</p> <p>Possible values are:</p> <p>GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN</p>
Speed	<p>Specifies the speed for the selected pins</p> <p>Possible values are:</p> <p>GPIO_SPEED_LOW GPIO_SPEED_MEDIUM GPIO_SPEED_FAST GPIO_SPEED_HIGH</p>
Alternate	<p>Peripheral to be connected to the selected pins.</p> <p>Possible values: GPIO_AFx PPP, where</p> <p>AFx: is the alternate function index</p> <p>PPP: is the peripheral instance</p> <p>Example: use GPIO_AF1_TIM1 to connect TIM1 IOs on AF1.</p> <p>These values are defined in the GPIO extended driver, since the AF mapping may change between product lines.</p> <p></p> <p>Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.</p>

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external

```
LEDsGPIO_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_FAST;
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 USART3 Tx (PC10, mapped on AF7) as alternate function:

```
GPIO_InitStruct.Pin = GPIO_PIN_10;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
GPIO_InitStruct.Alternate = GPIO_AF7_USART3;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

### 2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, `stm32f4xx_hal_cortex.c`, provides APIs to handle NVIC and Systick. The supported APIs include:

- HAL\_NVIC\_SetPriorityGrouping()
- HAL\_NVIC\_SetPriority()
- HAL\_NVIC\_EnableIRQ() / HAL\_NVIC\_DisableIRQ()
- HAL\_NVIC\_SystemReset()
- HAL\_NVIC\_GetPendingIRQ() / HAL\_NVIC\_SetPendingIRQ() / HAL\_NVIC\_ClearPendingIRQ()
- HAL\_SYSTICK\_Config()
- HAL\_SYSTICK\_CLKSourceConfig()

### 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\_PVDCfg()
  - 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()

Depending on the STM32 Series, extension functions are available in `stm32f4xx_hal_pwr_ex`. Here are a few examples (the list is not exhaustive)

- Backup domain registers enable/disable
  - HAL\_PWREx\_EnableBkUpReg() / HAL\_PWREx\_DisableBkUpReg()
- Flash overdrive control and flash power-down, for STM32F429/F439xx only
  - HAL\_PWREx\_ActivateOverDrive()
  - HAL\_PWREx\_EnableFlashPowerDown().

## 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: <code>#define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!&lt;External interrupt line 16 Connected to the PVD EXTI Line */</code>
_HAL_PPP_EXTI_ENABLE_IT(__EXTI_LINE__)	Enables a given EXTI line Example: <code>_HAL_PVD_EXTI_ENABLE_IT(PWR_EXTI_LINE_PVD)</code>
_HAL_PPP_EXTI_DISABLE_IT(__EXTI_LINE__)	Disables a given EXTI line. Example: <code>_HAL_PVD_EXTI_DISABLE_IT(PWR_EXTI_LINE_PVD)</code>
_HAL_PPP_EXTI_GET_FLAG(__EXTI_LINE__)	Gets a given EXTI line interrupt flag pending bit status. Example: <code>_HAL_PVD_EXTI_GET_FLAG(PWR_EXTI_LINE_PVD)</code>
_HAL_PPP_EXTI_CLEAR_FLAG(__EXTI_LINE__)	Clears a given EXTI line interrupt flag pending bit. Example: <code>_HAL_PVD_EXTI_CLEAR_FLAG(PWR_EXTI_LINE_PVD)</code>
_HAL_PPP_EXTI_GENERATE_SWIT(__EXTI_LINE__)	Generates a software interrupt for a given EXTI line. Example: <code>_HAL_PVD_EXTI_GENERATE_SWIT (PWR_EXTI_LINE_PVD)</code>

If the EXTI interrupt mode is selected, the user application must call HAL\_PPP\_FUNCTION\_IRQHandler() (for example HAL\_PWR\_PVD\_IRQHandler()), from stm32f4xx\_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 Stream (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 stream, 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
- Stream 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 configured. 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: enables the specified DMA Stream.
- \_\_HAL\_DMA\_DISABLE: disables the specified DMA Stream.
- \_\_HAL\_DMA\_GET\_FS: returns the current DMA Stream FIFO filled level.
- \_\_HAL\_DMA\_GET\_FLAG: gets the DMA Stream pending flags.
- \_\_HAL\_DMA\_CLEAR\_FLAG: clears the DMA Stream pending flags.
- \_\_HAL\_DMA\_ENABLE\_IT: enables the specified DMA Stream interrupts.
- \_\_HAL\_DMA\_DISABLE\_IT: disables the specified DMA Stream interrupts.
- \_\_HAL\_DMA\_GET\_IT\_SOURCE: checks whether the specified DMA stream interrupt has occurred or not.



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



DMA double-buffering feature is handled as an extension API.



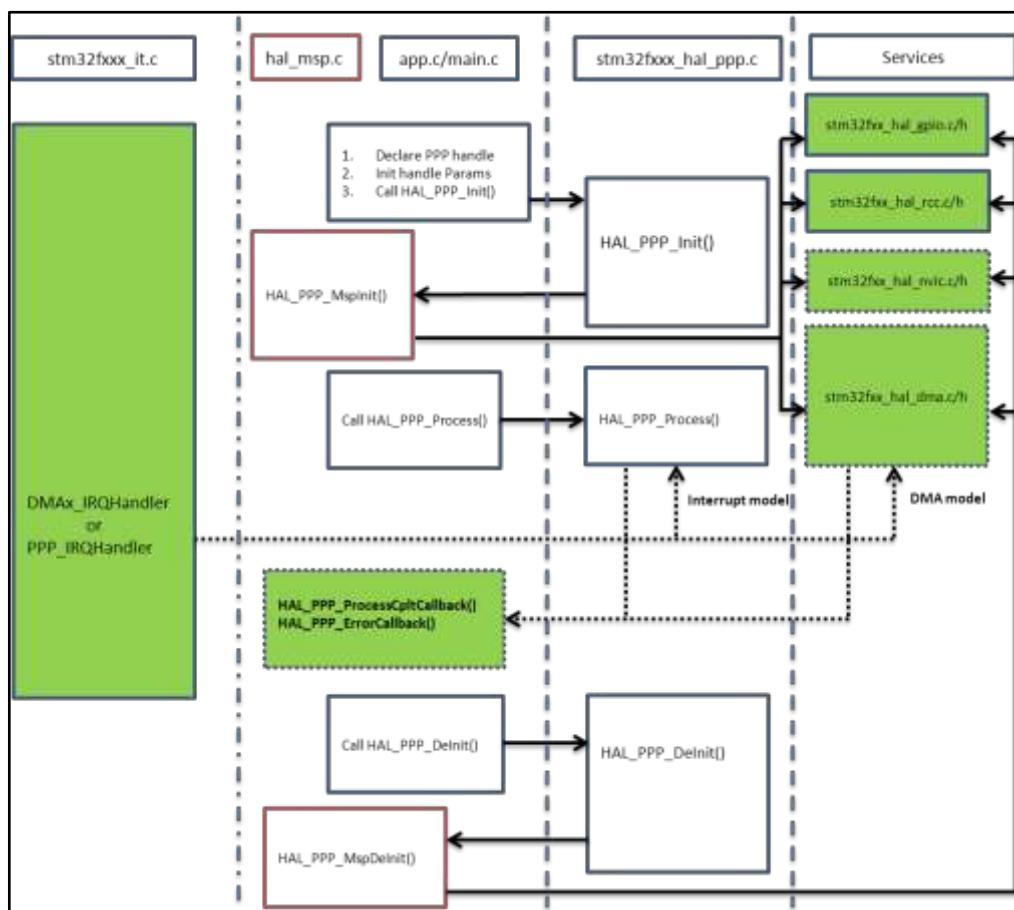
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

## 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



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 `stm32f4xx_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
  - Set priority grouping to 4 preemption bits
  - 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-Initializations.
- `HAL_GetTick()`: this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- `HAL_Delay()`: this function implements a delay (expressed in milliseconds) using the SysTick timer.

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



In STM32Cube V1.0 implemented in STM32CubeF2 and STM32CubeF4 first versions, the SysTick timer is used as default timebase. This has been modified to allow implementing user-defined timebases (such as a general-purpose timer), keeping in mind that the timebase duration must be kept at 1 ms since all `PPP_TIMEOUT_VALUES` are defined and handled in milliseconds. This enhancement is implemented in STM32Cube V1.1 that is deployed starting from STM32CubeL0/F0/F3 and later. This modification is backward compatible with STM32Cube V1.0 implementation. Functions affecting timebase configurations are declared as `__Weak` to allow different implementations in the user file.

### 2.12.2.2 HAL clock initialization

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

```
static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;
  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
```

```

RCC_OscInitStruct.HSEState = RCC_HSE_ON; RCC_OscInitStruct.PLL.PLLState =
RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 25; RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 7;
HAL_RCC_OscConfig(&RCC_OscInitStruct);
/* 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_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5); }

```

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

```

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

The HAL MSP callback is implemented inside the `stm32f4xx_hal_msp.c` file in the user folders. An `stm32f4xx_hal_msp.c` file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

`Stm32f4xx_hal_msp.c` file contains the following functions:

Table 14: MSP functions

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

Routine	Description
<b>void HAL_PPP_MspDelInit()</b>	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\_MspDelInit()*. In this case the *HAL\_PPP\_MspInit()* and *HAL\_PPP\_MspDelInit()* 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\_MspDelInit()* 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\_MspDelInit()*.

If there is nothing to be initialized by the global *HAL\_MspInit()* and *HAL\_MspDelInit()*, 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_t Size, uint32_t Timeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(...) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(...)

return HELIAC; }
```

### 2.12.3.2 Interrupt mode

In Interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed

in real-time about the process completion. The user can also get the process status through the *HAL\_PPP\_GetState()* function.

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

- *HAL\_PPP\_Process\_IT()*: launch the process
- *HAL\_PPP\_IRQHandler()*: the global PPP peripheral interruption
- *\_\_weak HAL\_PPP\_ProcCpltCallback ()*: the callback relative to the process completion.
- *\_\_weak HAL\_PPP\_ProcErrorCallback()*: 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 *stm32f4xx\_it.c*.

The *HAL\_PPP\_ProcCpltCallback()* 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 = USART3;
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)
{}
```

*stm32f4xx\_it.c* file:

```
extern UART_HandleTypeDef UartHandle;
void USART3_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 *stm32f4xx\_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 = UART3;
    HAL_UART_Init(&UartHandle);
    ...
}
void HAL_USART_MspInit (USART_HandleTypeDef * huart)
{
    static DMA_HandleTypeDef hdma_tx;
    static DMA_HandleTypeDef hdma_rx;
    ...
    __HAL_LINKDMA(UartHandle, DMA_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:

```
USART_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 = USART3;
    HAL_UART_Init(&UartHandle);
    HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
    while (1);
}
void HAL_UART_TxCpltCallback(USART_HandleTypeDef *phuart)
```

```
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{}
```

***stm32f4xx\_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 ... ( <i>HAL_MAX_DELAY</i> -1) <sup>(1)</sup>	Timeout in ms
<i>HAL_MAX_DELAY</i> <sup>(1)</sup>	Infinite poll till process is successful

**Notes:**

<sup>(1)</sup>*HAL\_MAX\_DELAY* is defined in the *stm32f4xx\_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_t
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 *stm32f34xx\_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 `stm32f4xx_hal_conf.h`:

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

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

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



**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 common driver

### 3.1 HAL Firmware driver API description

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

#### 3.1.1 How to use this driver

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

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

#### 3.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the Flash interface the NVIC allocation and initial clock configuration. It initializes the systick also when timeout is needed and the backup domain when enabled.
- de-Initialize common part of the HAL
- [\*\*HAL\\_Init\(\)\*\*](#)
- [\*\*HAL\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_MspDelInit\(\)\*\*](#)

#### 3.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- 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\\_GetHalVersion\(\)\*\*](#)
- [\*\*HAL\\_GetREVID\(\)\*\*](#)
- [\*\*HAL\\_GetDEVID\(\)\*\*](#)
- [\*\*HAL\\_EnableDBGSleepMode\(\)\*\*](#)
- [\*\*HAL\\_DisableDBGSleepMode\(\)\*\*](#)
- [\*\*HAL\\_EnableDBGStopMode\(\)\*\*](#)

- `HAL_DisableDBGStopMode()`
- `HAL_EnableDBGStandbyMode()`
- `HAL_DisableDBGStandbyMode()`
- `HAL_EnableCompensationCell()`
- `HAL_DisableCompensationCell()`
- `HAL_EnableMemorySwappingBank()`
- `HAL_DisableMemorySwappingBank()`

### 3.1.4 Initialization and de-initialization Functions

#### 3.1.4.1 HAL\_Init

Function Name	<b>HAL_StatusTypeDef HAL_Init ( void )</b>
Function Description	This function is used to initialize the HAL Library; it must be the first instruction to be executed in the main program (before to call any other HAL function), it performs the following: Configure the Flash prefetch, instruction and Data caches.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• SysTick is used as time base for the HAL_Delay() function, the application need to ensure that the SysTick time base is always set to 1 millisecond to have correct HAL operation.</li> </ul>

#### 3.1.4.2 HAL\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_DeInit ( void )</b>
Function Description	This function de-Initializes common part of the HAL and stops the systick.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 3.1.4.3 HAL\_MspInit

---

Function Name	<b>void HAL_MspInit ( void )</b>
Function Description	Initializes the MSP.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 3.1.4.4 HAL\_MspDeInit

Function Name	<b>void HAL_MspDeInit ( void )</b>
Function Description	Deinitializes the MSP.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 3.1.5 HAL Control functions

### 3.1.5.1 HAL\_IncTick

Function Name	<b>void HAL_IncTick ( void )</b>
Function Description	This function is called from SysTick ISR each 1 millisecond, to increment a global variable "uwTick" used as time base.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 3.1.5.2 HAL\_GetTick

---

Function Name	<b>uint32_t HAL_GetTick ( void )</b>
Function Description	Povides a tick value in millisecond.
Parameters	<ul style="list-style-type: none"> <li>None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>tick value</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 3.1.5.3 HAL\_Delay

Function Name	<b>void HAL_Delay ( __IO uint32_t Delay)</b>
Function Description	Provides a blocking delay in millisecond.
Parameters	<ul style="list-style-type: none"> <li><b>Delay</b> : specifies the delay time length, in milliseconds.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>Care must be taken when using HAL_Delay(), this function provides accurate delay (in milliseconds) based on 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. To change the SysTick interrupt priority you have to use HAL_NVIC_SetPriority() function.</li> </ul>

### 3.1.5.4 HAL\_GetHalVersion

Function Name	<b>uint32_t HAL_GetHalVersion ( void )</b>
Function Description	Returns the HAL revision.
Parameters	<ul style="list-style-type: none"> <li>None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>version</b> : 0xXYZR (8bits for each decimal, R for RC)</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 3.1.5.5 HAL\_GetREVID

Function Name	<b>uint32_t HAL_GetREVID ( void )</b>
Function Description	Returns the device revision identifier.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>Device revision identifier</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 3.1.5.6 HAL\_GetDEVID

Function Name	<b>uint32_t HAL_GetDEVID ( void )</b>
Function Description	Returns the device identifier.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>Device identifier</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 3.1.5.7 HAL\_EnableDBGSleepMode

Function Name	<b>void HAL_EnableDBGSleepMode ( void )</b>
Function Description	Enable the Debug Module during SLEEP mode.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 3.1.5.8 HAL\_DisableDBGSleepMode

Function Name	<b>void HAL_DisableDBGSleepMode ( void )</b>
Function Description	Disable the Debug Module during SLEEP mode.
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 3.1.5.9 HAL\_DisableDBGSleepMode

Function Name	<b>void HAL_DisableDBGStopMode ( void )</b>
Function Description	Enable the Debug Module during STOP mode.
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 3.1.5.10 HAL\_DisableDBGStopMode

Function Name	<b>void HAL_DisableDBGStopMode ( void )</b>
Function Description	Disable the Debug Module during STOP mode.
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 3.1.5.11 HAL\_EnableDBGStandbyMode

Function Name	<b>void HAL_EnableDBGStandbyMode ( void )</b>
---------------	---

---

Function Description	Enable the Debug Module during STANDBY mode.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 3.1.5.12 HAL\_DisableDBGStandbyMode

Function Name	<b>void HAL_DisableDBGStandbyMode ( void )</b>
Function Description	Disable the Debug Module during STANDBY mode.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 3.1.5.13 HAL\_EnableCompensationCell

Function Name	<b>void HAL_EnableCompensationCell ( void )</b>
Function Description	Enables the I/O Compensation Cell.
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.</li></ul>

### 3.1.5.14 HAL\_DisableCompensationCell

Function Name	<b>void HAL_DisableCompensationCell ( void )</b>
Function Description	Power-down the I/O Compensation Cell.
Return values	<ul style="list-style-type: none"><li>None.</li></ul>

## Notes

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

### 3.1.5.15 HAL\_EnableMemorySwappingBank

Function Name	<b>void HAL_EnableMemorySwappingBank ( void )</b>
Function Description	Enables the Internal FLASH Bank Swapping.
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• This function can be used only for STM32F42xxx/43xxx devices.</li><li>• Flash Bank2 mapped at 0x08000000 (and aliased @0x00000000) and Flash Bank1 mapped at 0x08100000 (and aliased at 0x00100000)</li></ul>

### 3.1.5.16 HAL\_DisableMemorySwappingBank

Function Name	<b>void HAL_DisableMemorySwappingBank ( void )</b>
Function Description	Disables the Internal FLASH Bank Swapping.
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• This function can be used only for STM32F42xxx/43xxx devices.</li><li>• The default state : Flash Bank1 mapped at 0x08000000 (and aliased @0x0000 0000) and Flash Bank2 mapped at 0x08100000 (and aliased at 0x00100000)</li></ul>

## 3.2 HAL Firmware driver defines

### 3.2.1 HAL

HAL

## 4 HAL ADC Generic Driver

### 4.1 ADC Firmware driver registers structures

#### 4.1.1 ADC\_HandleTypeDef

*ADC\_HandleTypeDef* is defined in the `stm32f4xx_hal_adc.h`

##### Data Fields

- *ADC\_TypeDef \* Instance*
- *ADC\_InitTypeDef Init*
- *\_\_IO uint32\_t NbrOfCurrentConversionRank*
- *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::NbrOfCurrentConversionRank*
  - ADC number of current conversion rank
- *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

#### 4.1.2 ADC\_InitTypeDef

*ADC\_InitTypeDef* is defined in the `stm32f4xx_hal_adc.h`

##### Data Fields

- *uint32\_t ClockPrescaler*
- *uint32\_t Resolution*
- *uint32\_t DataAlign*
- *uint32\_t ScanConvMode*
- *uint32\_t EOCSelection*
- *uint32\_t ContinuousConvMode*

- *uint32\_t DMAContinuousRequests*
- *uint32\_t NbrOfConversion*
- *uint32\_t DiscontinuousConvMode*
- *uint32\_t NbrOfDiscConversion*
- *uint32\_t ExternalTrigConvEdge*
- *uint32\_t ExternalTrigConv*

## Field Documentation

- *uint32\_t ADC\_InitTypeDef::ClockPrescaler*
  - Select the frequency of the clock to the ADC. The clock is common for all the ADCs. This parameter can be a value of [ADC\\_ClockPrescaler](#)
- *uint32\_t ADC\_InitTypeDef::Resolution*
  - Configures the ADC resolution dual mode. This parameter can be a value of [ADC\\_Resolution](#)
- *uint32\_t ADC\_InitTypeDef::DataAlign*
  - Specifies whether the ADC data alignment is left or right. This parameter can be a value of [ADC\\_data\\_align](#)
- *uint32\_t ADC\_InitTypeDef::ScanConvMode*
  - Specifies whether the conversion is performed in Scan (multi channels) or Single (one channel) mode. This parameter can be set to ENABLE or DISABLE
- *uint32\_t ADC\_InitTypeDef::EOCSelection*
  - Specifies whether the EOC flag is set at the end of single channel conversion or at the end of all conversions. This parameter can be a value of [ADC\\_EOCSelection](#)
- *uint32\_t ADC\_InitTypeDef::ContinuousConvMode*
  - Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to ENABLE or DISABLE.
- *uint32\_t ADC\_InitTypeDef::DMAContinuousRequests*
  - Specifies whether the DMA requests is performed in Continuous or in Single mode. This parameter can be set to ENABLE or DISABLE.
- *uint32\_t ADC\_InitTypeDef::NbrOfConversion*
  - Specifies the number of ADC conversions that will be done using the sequencer for regular channel group. This parameter must be a number between Min\_Data = 1 and Max\_Data = 16.
- *uint32\_t ADC\_InitTypeDef::DiscontinuousConvMode*
  - Specifies whether the conversion is performed in Discontinuous or not for regular channels. This parameter can be set to ENABLE or DISABLE.
- *uint32\_t ADC\_InitTypeDef::NbrOfDiscConversion*
  - Specifies the number of ADC discontinuous conversions that will be done using the sequencer for regular channel group. This parameter must be a number between Min\_Data = 1 and Max\_Data = 8.
- *uint32\_t ADC\_InitTypeDef::ExternalTrigConvEdge*
  - Select the external trigger edge and enable the trigger of a regular group. This parameter can be a value of [ADC\\_External\\_trigger\\_edge-Regular](#)
- *uint32\_t ADC\_InitTypeDef::ExternalTrigConv*
  - Select the external event used to trigger the start of conversion of a regular group. This parameter can be a value of [ADC\\_External\\_trigger\\_Source-Regular](#)

### 4.1.3 ADC\_ChannelConfTypeDef

*ADC\_ChannelConfTypeDef* is defined in the `stm32f4xx_hal_adc.h`

#### Data Fields

- *uint32\_t Channel*
- *uint32\_t Rank*
- *uint32\_t SamplingTime*
- *uint32\_t Offset*

#### Field Documentation

- ***uint32\_t ADC\_ChannelConfTypeDef::Channel***
  - The ADC channel to configure. This parameter can be a value of [\*ADC\\_channels\*](#)
- ***uint32\_t ADC\_ChannelConfTypeDef::Rank***
  - The rank in the regular group sequencer. This parameter must be a number between Min\_Data = 1 and Max\_Data = 16
- ***uint32\_t ADC\_ChannelConfTypeDef::SamplingTime***
  - The sample time value to be set for the selected channel. This parameter can be a value of [\*ADC\\_sampling\\_times\*](#)
- ***uint32\_t ADC\_ChannelConfTypeDef::Offset***
  - Reserved for future use, can be set to 0

### 4.1.4 ADC\_AnalogWDGConfTypeDef

*ADC\_AnalogWDGConfTypeDef* is defined in the `stm32f4xx_hal_adc.h`

#### Data Fields

- *uint32\_t WatchdogMode*
- *uint32\_t HighThreshold*
- *uint32\_t LowThreshold*
- *uint32\_t Channel*
- *uint32\_t ITMode*
- *uint32\_t WatchdogNumber*

#### Field Documentation

- ***uint32\_t ADC\_AnalogWDGConfTypeDef::WatchdogMode***
  - Configures the ADC analog watchdog mode. This parameter can be a value of [\*ADC\\_analog\\_watchdog\\_selection\*](#).
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::HighThreshold***
  - Configures the ADC analog watchdog High threshold value. This parameter must be a 12-bit value.
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::LowThreshold***

- Configures the ADC analog watchdog High threshold value. This parameter must be a 12-bit value.
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::Channel***
  - Configures ADC channel for the analog watchdog. This parameter has an effect only if watchdog mode is configured on single channel. This parameter can be a value of [ADC\\_channels](#).
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::ITMode***
  - Specifies whether the analog watchdog is configured in interrupt mode or in polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t ADC\_AnalogWDGConfTypeDef::WatchdogNumber***
  - Reserved for future use, can be set to 0

#### 4.1.5 ADC\_Common\_TypeDef

*ADC\_Common\_TypeDef* is defined in the stm32f439xx.h

##### Data Fields

- ***\_\_IO uint32\_t CSR***
- ***\_\_IO uint32\_t CCR***
- ***\_\_IO uint32\_t CDR***

##### Field Documentation

- ***\_\_IO uint32\_t ADC\_Common\_TypeDef::CSR***
  - ADC Common status register, Address offset: ADC1 base address + 0x300
- ***\_\_IO uint32\_t ADC\_Common\_TypeDef::CCR***
  - ADC common control register, Address offset: ADC1 base address + 0x304
- ***\_\_IO uint32\_t ADC\_Common\_TypeDef::CDR***
  - ADC common regular data register for dual AND triple modes, Address offset: ADC1 base address + 0x308

#### 4.1.6 ADC\_TypeDef

*ADC\_TypeDef* is defined in the stm32f439xx.h

##### Data Fields

- ***\_\_IO uint32\_t SR***
- ***\_\_IO uint32\_t CR1***
- ***\_\_IO uint32\_t CR2***
- ***\_\_IO uint32\_t SMPR1***
- ***\_\_IO uint32\_t SMPR2***
- ***\_\_IO uint32\_t JOFR1***
- ***\_\_IO uint32\_t JOFR2***
- ***\_\_IO uint32\_t JOFR3***
- ***\_\_IO uint32\_t JOFR4***
- ***\_\_IO uint32\_t HTR***
- ***\_\_IO uint32\_t LTR***

- `__IO uint32_t SQR1`
- `__IO uint32_t SQR2`
- `__IO uint32_t SQR3`
- `__IO uint32_t JSQR`
- `__IO uint32_t JDR1`
- `__IO uint32_t JDR2`
- `__IO uint32_t JDR3`
- `__IO uint32_t JDR4`
- `__IO uint32_t DR`

### Field Documentation

- `__IO uint32_t ADC_TypeDef::SR`  
– ADC status register, Address offset: 0x00
- `__IO uint32_t ADC_TypeDef::CR1`  
– ADC control register 1, Address offset: 0x04
- `__IO uint32_t ADC_TypeDef::CR2`  
– ADC control register 2, Address offset: 0x08
- `__IO uint32_t ADC_TypeDef::SMPR1`  
– ADC sample time register 1, Address offset: 0x0C
- `__IO uint32_t ADC_TypeDef::SMPR2`  
– ADC sample time register 2, Address offset: 0x10
- `__IO uint32_t ADC_TypeDef::JOFR1`  
– ADC injected channel data offset register 1, Address offset: 0x14
- `__IO uint32_t ADC_TypeDef::JOFR2`  
– ADC injected channel data offset register 2, Address offset: 0x18
- `__IO uint32_t ADC_TypeDef::JOFR3`  
– ADC injected channel data offset register 3, Address offset: 0x1C
- `__IO uint32_t ADC_TypeDef::JOFR4`  
– ADC injected channel data offset register 4, Address offset: 0x20
- `__IO uint32_t ADC_TypeDef::HTR`  
– ADC watchdog higher threshold register, Address offset: 0x24
- `__IO uint32_t ADC_TypeDef::LTR`  
– ADC watchdog lower threshold register, Address offset: 0x28
- `__IO uint32_t ADC_TypeDef::SQR1`  
– ADC regular sequence register 1, Address offset: 0x2C
- `__IO uint32_t ADC_TypeDef::SQR2`  
– ADC regular sequence register 2, Address offset: 0x30
- `__IO uint32_t ADC_TypeDef::SQR3`  
– ADC regular sequence register 3, Address offset: 0x34
- `__IO uint32_t ADC_TypeDef::JSQR`  
– ADC injected sequence register, Address offset: 0x38
- `__IO uint32_t ADC_TypeDef::JDR1`  
– ADC injected data register 1, Address offset: 0x3C
- `__IO uint32_t ADC_TypeDef::JDR2`  
– ADC injected data register 2, Address offset: 0x40
- `__IO uint32_t ADC_TypeDef::JDR3`  
– ADC injected data register 3, Address offset: 0x44
- `__IO uint32_t ADC_TypeDef::JDR4`  
– ADC injected data register 4, Address offset: 0x48
- `__IO uint32_t ADC_TypeDef::DR`

- ADC regular data register, Address offset: 0x4C

## 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.
2. Interrupt generation at the end of 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 x.
5. Data alignment with in-built data coherency.
6. Channel-wise programmable sampling time.
7. External trigger option with configurable polarity for both regular and injected conversion.
8. Dual/Triple mode (on devices with 2 ADCs or more).
9. Configurable DMA data storage in Dual/Triple ADC mode.
10. Configurable delay between conversions in Dual/Triple interleaved mode.
11. ADC conversion type (refer to the datasheets).
12. ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
13. ADC input range: VREF(minus) = VIN = VREF(plus).
14. DMA request generation during regular channel conversion.

### 4.2.2 How to use this driver

1. Initialize the ADC low level resources by implementing the HAL\_ADC\_MspInit():
  - a. Enable the ADC interface clock using \_\_ADC\_CLK\_ENABLE()
  - b. ADC pins configuration
    - Enable the clock for the ADC GPIOs using the following function:  
  \_\_GPIOx\_CLK\_ENABLE()
    - Configure these ADC pins in analog mode using HAL\_GPIO\_Init()
  - c. In case of using interrupts (e.g. HAL\_ADC\_Start\_IT())
    - Configure the ADC interrupt priority using HAL\_NVIC\_SetPriority()
    - Enable the ADC IRQ handler using HAL\_NVIC\_EnableIRQ()
    - In ADC IRQ handler, call HAL\_ADC\_IRQHandler()
  - d. In case of using DMA to control data transfer (e.g. HAL\_ADC\_Start\_DMA())
    - Enable the DMAx interface clock using \_\_DMAx\_CLK\_ENABLE()
    - Configure and enable two DMA streams stream for managing data transfer from peripheral to memory (output stream)
    - Associate the initialized DMA handle to the CRYP DMA handle using  
  \_\_HAL\_LINKDMA()
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream.

2. Configure the ADC Prescaler, conversion resolution and data alignment using the HAL\_ADC\_Init() function.
3. Configure the ADC regular channels group features, use HAL\_ADC\_Init() and HAL\_ADC\_ConfigChannel() functions.
4. Three operation modes are available within this driver :

### Polling mode IO operation

- Start the ADC peripheral using HAL\_ADC\_Start()
- Wait for end of conversion using HAL\_ADC\_PollForConversion(), at this stage user can specify the value of timeout according to his end application
- To read the ADC converted values, use the HAL\_ADC\_GetValue() function.
- Stop the ADC peripheral using HAL\_ADC\_Stop()

### Interrupt mode IO operation

- Start the ADC peripheral using HAL\_ADC\_Start\_IT()
- Use HAL\_ADC\_IRQHandler() called under ADC\_IRQHandler() Interrupt subroutine
- At ADC end of conversion HAL\_ADC\_ConvCpltCallback() function is executed and user can add his own code by customization of function pointer HAL\_ADC\_ConvCpltCallback
- In case of ADC Error, HAL\_ADC\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_ADC\_ErrorCallback
- Stop the ADC peripheral using HAL\_ADC\_Stop\_IT()

### DMA mode IO operation

- Start the ADC peripheral using HAL\_ADC\_Start\_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion
- At The end of data transfer by HAL\_ADC\_ConvCpltCallback() function is executed and user can add his own code by customization of function pointer HAL\_ADC\_ConvCpltCallback
- In case of transfer Error, HAL\_ADC\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_ADC\_ErrorCallback
- Stop the ADC peripheral using HAL\_ADC\_Stop\_DMA()

### ADC HAL driver macros list

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

- \_\_HAL\_ADC\_ENABLE : Enable the ADC peripheral
- \_\_HAL\_ADC\_DISABLE : Disable the ADC peripheral
- \_\_HAL\_ADC\_ENABLE\_IT: Enable the ADC end of conversion interrupt
- \_\_HAL\_ADC\_DISABLE\_IT: Disable the ADC end of conversion interrupt
- \_\_HAL\_ADC\_GET\_IT\_SOURCE: Check if the specified ADC interrupt source is enabled or disabled
- \_\_HAL\_ADC\_CLEAR\_FLAG: Clear the ADC's pending flags
- \_\_HAL\_ADC\_GET\_FLAG: Get the selected ADC's flag status
- \_\_HAL\_ADC\_GET\_RESOLUTION: Return resolution bits in CR1 register



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

### 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\\_DelInit\(\)\*](#)
- [\*HAL\\_ADC\\_MspInit\(\)\*](#)
- [\*HAL\\_ADC\\_MspDelInit\(\)\*](#)

### 4.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion of regular channel.
- Stop conversion of regular channel.
- Start conversion of regular channel and enable interrupt.
- Stop conversion of regular channel and disable interrupt.
- Start conversion of regular channel and enable DMA transfer.
- Stop conversion of regular channel and disable DMA transfer.
- handle ADC interrupt request.
- [\*HAL\\_ADC\\_Start\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\(\)\*](#)
- [\*HAL\\_ADC\\_PollForConversion\(\)\*](#)
- [\*HAL\\_ADC\\_PollForEvent\(\)\*](#)
- [\*HAL\\_ADC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_ADC\\_IRQHandler\(\)\*](#)
- [\*HAL\\_ADC\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\\_DMA\(\)\*](#)
- [\*HAL\\_ADC\\_GetValue\(\)\*](#)
- [\*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 regular channels.
- Configure injected channels.
- Configure multimode.
- Configure the analog watch dog.
- [\*HAL\\_ADC\\_ConfigChannel\(\)\*](#)

- [\*\*\*HAL\\_ADC\\_AnalogWDGConfig\(\)\*\*\*](#)

## 4.2.6 Peripheral State and errors functions

This subsection provides functions allowing to

- Check the ADC state
- Check the ADC Error
- [\*\*\*HAL\\_ADC\\_GetState\(\)\*\*\*](#)
- [\*\*\*HAL\\_ADC\\_GetError\(\)\*\*\*](#)

## 4.2.7 Initialization and de-initialization functions

### 4.2.7.1 HAL\_ADC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Init ( <a href="#"><b><i>ADC_HandleTypeDef</i></b></a> *hadc)</b>
Function Description	Initializes the ADCx peripheral according to the specified parameters in the <a href="#"><b><i>ADC_InitStruct</i></b></a> and initializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a <a href="#"><b><i>ADC_HandleTypeDef</i></b></a> structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function is used to configure the global features of the ADC ( ClockPrescaler, Resolution, Data Alignment and number of conversion), however, the rest of the configuration parameters are specific to the regular channels group (scan mode activation, continuous mode activation, External trigger source and edge, DMA continuous request after the last transfer and End of conversion selection).</li> </ul>

### 4.2.7.2 HAL\_ADC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_ADC_DeInit ( <a href="#"><b><i>ADC_HandleTypeDef</i></b></a> *hadc)</b>
Function Description	Deinitializes the ADCx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a <a href="#"><b><i>ADC_HandleTypeDef</i></b></a> structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.7.3 HAL\_ADC\_MspInit

Function Name	<b>void HAL_ADC_MspInit ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Initializes the ADC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 4.2.7.4 HAL\_ADC\_MspDeInit

Function Name	<b>void HAL_ADC_MspDeInit ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Deinitializes the ADC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 4.2.8 IO operation functions

#### 4.2.8.1 HAL\_ADC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Enables ADC and starts conversion of the regular channels.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li></ul>

---

Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 4.2.8.2 HAL\_ADC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Disables ADC and stop conversion of regular channels.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• Caution: This function will stop also injected channels.</li></ul>

#### 4.2.8.3 HAL\_ADC\_PollForConversion

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForConversion ( <i>ADC_HandleTypeDef</i> * hadc, uint32_t Timeout)</b>
Function Description	Poll for regular conversion complete.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li><li>• <b>Timeout</b> : Timeout value in millisecond.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 4.2.8.4 HAL\_ADC\_PollForEvent

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForEvent ( <i>ADC_HandleTypeDef</i> * hadc, uint32_t EventType, uint32_t Timeout)</b>
---------------	--

Function Description	Poll for conversion event.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> <li>• <b>EventType</b> : the ADC event type. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>AWD_EVENT</b> : ADC Analog watch Dog event.</li> <li>– <b>OVR_EVENT</b> : ADC Overrun event.</li> </ul> </li> <li>• <b>Timeout</b> : Timeout value in millisecond.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.8.5 HAL\_ADC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start_IT (</b> <b><i>ADC_HandleTypeDef * hadc</i></b> )
Function Description	Enables the interrupt and starts ADC conversion of regular channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b>.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.8.6 HAL\_ADC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop_IT (</b> <b><i>ADC_HandleTypeDef * hadc</i></b> )
Function Description	Disables the interrupt and stop ADC conversion of regular channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b>.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Caution: This function will stop also injected channels.</li> </ul>

#### 4.2.8.7 HAL\_ADC\_IRQHandler

Function Name	<b>void HAL_ADC_IRQHandler ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Handles ADC interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.8.8 HAL\_ADC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start_DMA ( <i>ADC_HandleTypeDef</i> * hadc, uint32_t * pData, uint32_t Length)</b>
Function Description	Enables ADC DMA request after last transfer (Single-ADC mode) and enables ADC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> <li>• <b>pData</b> : The destination Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from ADC peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.8.9 HAL\_ADC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop_DMA ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Disables ADC DMA (Single-ADC mode) and disables ADC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified ADC.

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul> |
| Notes         | <ul style="list-style-type: none"> <li>• None.</li> </ul>             |

#### 4.2.8.10 HAL\_ADC\_GetValue

Function Name	<code>uint32_t HAL_ADC_GetValue ( <i>ADC_HandleTypeDef</i> * hadc)</code>
Function Description	Gets the converted value from data register of regular channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a <i>ADC_HandleTypeDef</i> structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Converted value</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.8.11 HAL\_ADC\_ConvCpltCallback

Function Name	<code>void HAL_ADC_ConvCpltCallback ( <i>ADC_HandleTypeDef</i> * hadc)</code>
Function Description	Regular conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a <i>ADC_HandleTypeDef</i> structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.8.12 HAL\_ADC\_ConvHalfCpltCallback

Function Name	<code>void HAL_ADC_ConvHalfCpltCallback ( <i>ADC_HandleTypeDef</i> * hadc)</code>
Function Description	Regular conversion half DMA transfer callback in non blocking

mode.

Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 4.2.8.13 HAL\_ADC\_LevelOutOfWindowCallback

Function Name	<b>void HAL_ADC_LevelOutOfWindowCallback ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Analog watchdog callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 4.2.8.14 HAL\_ADC\_ErrorCallback

Function Name	<b>void HAL_ADC_ErrorCallback ( <i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Error ADC callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 4.2.9 Peripheral Control functions

#### 4.2.9.1 HAL\_ADC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_ADC_ConfigChannel (</b> <b>ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef *</b> <b>sConfig)</b>
Function Description	Configures for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> <li>• <b>sConfig</b> : ADC configuration structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 4.2.9.2 HAL\_ADC\_AnalogWDGConfig

Function Name	<b>HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (</b> <b>ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef *</b> <b>AnalogWDGConfig)</b>
Function Description	Configures the analog watchdog.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> <li>• <b>AnalogWDGConfig</b> : pointer to an ADC_AnalogWDGConfTypeDef structure that contains the configuration information of ADC analog watchdog.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 4.2.10 ADC Peripheral State functions

#### 4.2.10.1 HAL\_ADC\_GetState

Function Name	<b>HAL_ADC_StateTypeDef HAL_ADC_GetState (</b> <b>ADC_HandleTypeDef * hadc)</b>
Function Description	return the ADC state
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified ADC.

- |               |  |
|---------------|--|
| Return values | <ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>            |

#### 4.2.10.2 HAL\_ADC\_GetError

Function Name	<code>uint32_t HAL_ADC_GetError ( ADC_HandleTypeDef * hadc)</code>
Function Description	Return the ADC error code.
Parameters	<ul style="list-style-type: none"><li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>ADC Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 4.3 ADC Firmware driver defines

### 4.3.1 ADC

ADC

*ADC\_analog\_watchdog\_selection*

- `#define: ADC_ANALOGWATCHDOG_SINGLE_REG  
((uint32_t)(ADC_CR1_AWDSDL | ADC_CR1_AWDEN))`
- `#define: ADC_ANALOGWATCHDOG_SINGLE_INJEC  
((uint32_t)(ADC_CR1_AWDSDL | ADC_CR1_JAWDEN))`
- `#define: ADC_ANALOGWATCHDOG_SINGLE_REGINJEC  
((uint32_t)(ADC_CR1_AWDSDL | ADC_CR1_AWDEN | ADC_CR1_JAWDEN))`
- `#define: ADC_ANALOGWATCHDOG_ALL_REG ((uint32_t)ADC_CR1_AWDEN)`

- #define: ***ADC\_ANALOGWATCHDOG\_ALL\_INJEC*** ((*uint32\_t*)***ADC\_CR1\_JAWDEN***)
  
- #define: ***ADC\_ANALOGWATCHDOG\_ALL\_REGINJEC*** ((*uint32\_t*)(***ADC\_CR1\_AWDEN*** | ***ADC\_CR1\_JAWDEN***))
  
- #define: ***ADC\_ANALOGWATCHDOG\_NONE*** ((*uint32\_t*)0x00000000)

#### ***ADC\_channels***

- #define: ***ADC\_CHANNEL\_0*** ((*uint32\_t*)0x00000000)
  
- #define: ***ADC\_CHANNEL\_1*** ((*uint32\_t*)***ADC\_CR1\_AWDCH\_0***)
  
- #define: ***ADC\_CHANNEL\_2*** ((*uint32\_t*)***ADC\_CR1\_AWDCH\_1***)
  
- #define: ***ADC\_CHANNEL\_3*** ((*uint32\_t*)(***ADC\_CR1\_AWDCH\_1*** | ***ADC\_CR1\_AWDCH\_0***))
  
- #define: ***ADC\_CHANNEL\_4*** ((*uint32\_t*)***ADC\_CR1\_AWDCH\_2***)
  
- #define: ***ADC\_CHANNEL\_5*** ((*uint32\_t*)(***ADC\_CR1\_AWDCH\_2*** | ***ADC\_CR1\_AWDCH\_0***))
  
- #define: ***ADC\_CHANNEL\_6*** ((*uint32\_t*)(***ADC\_CR1\_AWDCH\_2*** | ***ADC\_CR1\_AWDCH\_1***))
  
- #define: ***ADC\_CHANNEL\_7*** ((*uint32\_t*)(***ADC\_CR1\_AWDCH\_2*** | ***ADC\_CR1\_AWDCH\_1*** | ***ADC\_CR1\_AWDCH\_0***))

- #define: ***ADC\_CHANNEL\_8 ((uint32\_t)ADC\_CR1\_AWDCH\_3)***
- #define: ***ADC\_CHANNEL\_9 ((uint32\_t)(ADC\_CR1\_AWDCH\_3 | ADC\_CR1\_AWDCH\_0))***
- #define: ***ADC\_CHANNEL\_10 ((uint32\_t)(ADC\_CR1\_AWDCH\_3 | ADC\_CR1\_AWDCH\_1))***
- #define: ***ADC\_CHANNEL\_11 ((uint32\_t)(ADC\_CR1\_AWDCH\_3 | ADC\_CR1\_AWDCH\_1 | ADC\_CR1\_AWDCH\_0))***
- #define: ***ADC\_CHANNEL\_12 ((uint32\_t)(ADC\_CR1\_AWDCH\_3 | ADC\_CR1\_AWDCH\_2))***
- #define: ***ADC\_CHANNEL\_13 ((uint32\_t)(ADC\_CR1\_AWDCH\_3 | ADC\_CR1\_AWDCH\_2 | ADC\_CR1\_AWDCH\_0))***
- #define: ***ADC\_CHANNEL\_14 ((uint32\_t)(ADC\_CR1\_AWDCH\_3 | ADC\_CR1\_AWDCH\_2 | ADC\_CR1\_AWDCH\_1))***
- #define: ***ADC\_CHANNEL\_15 ((uint32\_t)(ADC\_CR1\_AWDCH\_3 | ADC\_CR1\_AWDCH\_2 | ADC\_CR1\_AWDCH\_1 | ADC\_CR1\_AWDCH\_0))***
- #define: ***ADC\_CHANNEL\_16 ((uint32\_t)ADC\_CR1\_AWDCH\_4)***
- #define: ***ADC\_CHANNEL\_17 ((uint32\_t)(ADC\_CR1\_AWDCH\_4 | ADC\_CR1\_AWDCH\_0))***

- #define: **ADC\_CHANNEL\_18** ((*uint32\_t*)(**ADC\_CR1\_AWDCH\_4** | **ADC\_CR1\_AWDCH\_1**))
- #define: **ADC\_CHANNEL\_TEMPSENSOR** ((*uint32\_t*)**ADC\_CHANNEL\_16**)
- #define: **ADC\_CHANNEL\_VREFINT** ((*uint32\_t*)**ADC\_CHANNEL\_17**)
- #define: **ADC\_CHANNEL\_VBAT** ((*uint32\_t*)**ADC\_CHANNEL\_18**)

#### ***ADC\_channels\_type***

- #define: **ALL\_CHANNELS** ((*uint32\_t*)0x00000001)
- #define: **REGULAR\_CHANNELS** ((*uint32\_t*)0x00000002)  
*reserved for future use*
- #define: **INJECTED\_CHANNELS** ((*uint32\_t*)0x00000003)  
*reserved for future use*

#### ***ADC\_ClockPrescaler***

- #define: **ADC\_CLOCKPRESCALER\_PCLK\_DIV2** ((*uint32\_t*)0x00000000)
- #define: **ADC\_CLOCKPRESCALER\_PCLK\_DIV4** ((*uint32\_t*)**ADC\_CCR\_ADCPRE\_0**)
- #define: **ADC\_CLOCKPRESCALER\_PCLK\_DIV6** ((*uint32\_t*)**ADC\_CCR\_ADCPRE\_1**)
- #define: **ADC\_CLOCKPRESCALER\_PCLK\_DIV8** ((*uint32\_t*)**ADC\_CCR\_ADCPRE**)

***ADC\_data\_align***

- #define: ***ADC\_DATAALIGN\_RIGHT ((uint32\_t)0x00000000)***
  
- #define: ***ADC\_DATAALIGN\_LEFT ((uint32\_t)ADC\_CR2\_ALIGN)***

***ADC\_EOCSelection***

- #define: ***EOC\_SEQ\_CONV ((uint32\_t)0x00000000)***
  
- #define: ***EOC\_SINGLE\_CONV ((uint32\_t)0x00000001)***
  
- #define: ***EOC\_SINGLE\_SEQ\_CONV ((uint32\_t)0x00000002)***  
*reserved for future use*

***ADC\_Error\_Code***

- #define: ***HAL\_ADC\_ERROR\_NONE ((uint32\_t)0x00)***  
*No error*
  
- #define: ***HAL\_ADC\_ERROR\_OVR ((uint32\_t)0x01)***  
*OVR error*
  
- #define: ***HAL\_ADC\_ERROR\_DMA ((uint32\_t)0x02)***  
*DMA transfer error*

***ADC\_Event\_type***

- #define: ***AWD\_EVENT ((uint32\_t)ADC\_FLAG\_AWD)***
  
- #define: ***OVR\_EVENT ((uint32\_t)ADC\_FLAG\_OVR)***

*ADC\_External\_trigger\_edge-Regular*

- #define: **ADC\_EXTERNALTRIGCONVEDGE\_NONE** ((*uint32\_t*)0x00000000)
  
- #define: **ADC\_EXTERNALTRIGCONVEDGE\_RISING** ((*uint32\_t*)**ADC\_CR2\_EXTEN\_0**)
  
- #define: **ADC\_EXTERNALTRIGCONVEDGE\_FALLING** ((*uint32\_t*)**ADC\_CR2\_EXTEN\_1**)
  
- #define: **ADC\_EXTERNALTRIGCONVEDGE\_RISINGFALLING** ((*uint32\_t*)**ADC\_CR2\_EXTEN**)

*ADC\_External\_trigger\_Source-Regular*

- #define: **ADC\_EXTERNALTRIGCONV\_T1\_CC1** ((*uint32\_t*)0x00000000)
  
- #define: **ADC\_EXTERNALTRIGCONV\_T1\_CC2** ((*uint32\_t*)**ADC\_CR2\_EXTSEL\_0**)
  
- #define: **ADC\_EXTERNALTRIGCONV\_T1\_CC3** ((*uint32\_t*)**ADC\_CR2\_EXTSEL\_1**)
  
- #define: **ADC\_EXTERNALTRIGCONV\_T2\_CC2** ((*uint32\_t*)(**ADC\_CR2\_EXTSEL\_1** | **ADC\_CR2\_EXTSEL\_0**))
  
- #define: **ADC\_EXTERNALTRIGCONV\_T2\_CC3** ((*uint32\_t*)**ADC\_CR2\_EXTSEL\_2**)
  
- #define: **ADC\_EXTERNALTRIGCONV\_T2\_CC4** ((*uint32\_t*)(**ADC\_CR2\_EXTSEL\_2** | **ADC\_CR2\_EXTSEL\_0**))

- #define: ***ADC\_EXTERNALTRIGCONV\_T2\_TRGO***  
((*uint32\_t*)(***ADC\_CR2\_EXTSEL\_2 | ADC\_CR2\_EXTSEL\_1***))
  
- #define: ***ADC\_EXTERNALTRIGCONV\_T3\_CC1*** ((*uint32\_t*)(***ADC\_CR2\_EXTSEL\_2 | ADC\_CR2\_EXTSEL\_1 | ADC\_CR2\_EXTSEL\_0***))
  
- #define: ***ADC\_EXTERNALTRIGCONV\_T3\_TRGO***  
((*uint32\_t*)***ADC\_CR2\_EXTSEL\_3***)
  
- #define: ***ADC\_EXTERNALTRIGCONV\_T4\_CC4*** ((*uint32\_t*)(***ADC\_CR2\_EXTSEL\_3 | ADC\_CR2\_EXTSEL\_0***))
  
- #define: ***ADC\_EXTERNALTRIGCONV\_T5\_CC1*** ((*uint32\_t*)(***ADC\_CR2\_EXTSEL\_3 | ADC\_CR2\_EXTSEL\_1***))
  
- #define: ***ADC\_EXTERNALTRIGCONV\_T5\_CC2*** ((*uint32\_t*)(***ADC\_CR2\_EXTSEL\_3 | ADC\_CR2\_EXTSEL\_1 | ADC\_CR2\_EXTSEL\_0***))
  
- #define: ***ADC\_EXTERNALTRIGCONV\_T5\_CC3*** ((*uint32\_t*)(***ADC\_CR2\_EXTSEL\_3 | ADC\_CR2\_EXTSEL\_2***))
  
- #define: ***ADC\_EXTERNALTRIGCONV\_T8\_CC1*** ((*uint32\_t*)(***ADC\_CR2\_EXTSEL\_3 | ADC\_CR2\_EXTSEL\_2 | ADC\_CR2\_EXTSEL\_0***))
  
- #define: ***ADC\_EXTERNALTRIGCONV\_T8\_TRGO***  
((*uint32\_t*)(***ADC\_CR2\_EXTSEL\_3 | ADC\_CR2\_EXTSEL\_2 | ADC\_CR2\_EXTSEL\_1***))
  
- #define: ***ADC\_EXTERNALTRIGCONV\_Ext\_IT11*** ((*uint32\_t*)***ADC\_CR2\_EXTSEL***)

***ADC\_flags\_definition***

- #define: ***ADC\_FLAG\_AWD ((uint32\_t)ADC\_SR\_AWD)***
- #define: ***ADC\_FLAG\_EOC ((uint32\_t)ADC\_SR\_EOC)***
- #define: ***ADC\_FLAG\_JEOC ((uint32\_t)ADC\_SR\_JEOC)***
- #define: ***ADC\_FLAG\_JSTRT ((uint32\_t)ADC\_SR\_JSTRT)***
- #define: ***ADC\_FLAG\_STRT ((uint32\_t)ADC\_SR\_STRT)***
- #define: ***ADC\_FLAG\_OVR ((uint32\_t)ADC\_SR\_OVR)***

***ADC\_interrupts\_definition***

- #define: ***ADC\_IT\_EOC ((uint32\_t)ADC\_CR1\_EOCIE)***
- #define: ***ADC\_IT\_AWD ((uint32\_t)ADC\_CR1\_AWDIE)***
- #define: ***ADC\_IT\_JEOC ((uint32\_t)ADC\_CR1\_JEOCIE)***
- #define: ***ADC\_IT\_OVR ((uint32\_t)ADC\_CR1\_OVRIE)***

***ADC\_Resolution***

- #define: ***ADC\_RESOLUTION12b ((uint32\_t)0x00000000)***

- #define: **ADC\_RESOLUTION10b** ((*uint32\_t*)**ADC\_CR1\_RES\_0**)
  - #define: **ADC\_RESOLUTION8b** ((*uint32\_t*)**ADC\_CR1\_RES\_1**)
  - #define: **ADC\_RESOLUTION6b** ((*uint32\_t*)**ADC\_CR1\_RES**)
- 
- ADC\_sampling\_times***
- #define: **ADC\_SAMPLETIME\_3CYCLES** ((*uint32\_t*)0x00000000)
  - #define: **ADC\_SAMPLETIME\_15CYCLES** ((*uint32\_t*)**ADC\_SMPR1\_SMP10\_0**)
  - #define: **ADC\_SAMPLETIME\_28CYCLES** ((*uint32\_t*)**ADC\_SMPR1\_SMP10\_1**)
  - #define: **ADC\_SAMPLETIME\_56CYCLES** ((*uint32\_t*)(**ADC\_SMPR1\_SMP10\_1** | **ADC\_SMPR1\_SMP10\_0**))
  - #define: **ADC\_SAMPLETIME\_84CYCLES** ((*uint32\_t*)**ADC\_SMPR1\_SMP10\_2**)
  - #define: **ADC\_SAMPLETIME\_112CYCLES** ((*uint32\_t*)(**ADC\_SMPR1\_SMP10\_2** | **ADC\_SMPR1\_SMP10\_0**))
  - #define: **ADC\_SAMPLETIME\_144CYCLES** ((*uint32\_t*)(**ADC\_SMPR1\_SMP10\_2** | **ADC\_SMPR1\_SMP10\_1**))
  - #define: **ADC\_SAMPLETIME\_480CYCLES** ((*uint32\_t*)**ADC\_SMPR1\_SMP10**)

## 5 HAL ADC Extension Driver

### 5.1 ADCEx Firmware driver registers structures

#### 5.1.1 ADC\_InjectionConfTypeDef

*ADC\_InjectionConfTypeDef* is defined in the `stm32f4xx_hal_adc_ex.h`

##### Data Fields

- `uint32_t InjectedChannel`
- `uint32_t InjectedRank`
- `uint32_t InjectedSamplingTime`
- `uint32_t InjectedOffset`
- `uint32_t InjectedNbrOfConversion`
- `uint32_t AutoInjectedConv`
- `uint32_t InjectedDiscontinuousConvMode`
- `uint32_t ExternalTrigInjecConvEdge`
- `uint32_t ExternalTrigInjecConv`

##### Field Documentation

- `uint32_t ADC_InjectionConfTypeDef::InjectedChannel`
  - Configure the ADC injected channel. This parameter can be a value of [ADC\\_channels](#)
- `uint32_t ADC_InjectionConfTypeDef::InjectedRank`
  - The rank in the injected group sequencer. This parameter must be a number between Min\_Data = 1 and Max\_Data = 4.
- `uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime`
  - The sample time value to be set for the selected channel. This parameter can be a value of [ADC\\_sampling\\_times](#)
- `uint32_t ADC_InjectionConfTypeDef::InjectedOffset`
  - Defines the offset to be subtracted from the raw converted data when convert injected channels. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF.
- `uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion`
  - Specifies the number of ADC conversions that will be done using the sequencer for injected channel group. This parameter must be a number between Min\_Data = 1 and Max\_Data = 4.
- `uint32_t ADC_InjectionConfTypeDef::AutoInjectedConv`
  - Enables or disables the selected ADC automatic injected group conversion after regular one
- `uint32_t ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode`
  - Specifies whether the conversion is performed in Discontinuous mode or not for injected channels. This parameter can be set to ENABLE or DISABLE.
- `uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge`
  - Select the external trigger edge and enable the trigger of an injected channels. This parameter can be a value of [ADCEx\\_External\\_trigger\\_edge\\_Injected](#)
- `uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv`

- Select the external event used to trigger the start of conversion of a injected channels. This parameter can be a value of [\*\*ADCEx\\_External\\_trigger\\_Source\\_Injected\*\*](#)

### 5.1.2 ADC\_MultiModeTypeDef

**ADC\_MultiModeTypeDef** is defined in the `stm32f4xx_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 Direct memory access mode for multi ADC mode. This parameter can be a value of [\*\*ADCEx\\_Direct\\_memory\\_access\\_mode\\_for\\_multi\\_mode\*\*](#)
- **`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\*\*](#)

## 5.2 ADCEx Firmware driver API description

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

### 5.2.1 How to use this driver

1. Initialize the ADC low level resources by implementing the `HAL_ADC_MspInit()`:
  - a. Enable the ADC interface clock using `__ADC_CLK_ENABLE()`
  - b. ADC pins configuration
    - Enable the clock for the ADC GPIOs using the following function:  
`__GPIOx_CLK_ENABLE()`
    - Configure these ADC pins in analog mode using `HAL_GPIO_Init()`
  - c. In case of using interrupts (e.g. `HAL_ADC_Start_IT()`)
    - Configure the ADC interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the ADC IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In ADC IRQ handler, call `HAL_ADC_IRQHandler()`
  - d. In case of using DMA to control data transfer (e.g. `HAL_ADC_Start_DMA()`)
    - Enable the DMAx interface clock using `__DMAx_CLK_ENABLE()`
    - Configure and enable two DMA streams stream for managing data transfer from peripheral to memory (output stream)

- Associate the initialized DMA handle to the ADC DMA handle using `__HAL_LINKDMA()`
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream.
2. Configure the ADC Prescaler, conversion resolution and data alignment using the `HAL_ADC_Init()` function.
  3. Configure the ADC Injected channels group features, use `HAL_ADC_Init()` and `HAL_ADC_ConfigChannel()` functions.
  4. Three operation modes are available within this driver :

### **Polling mode IO operation**

- Start the ADC peripheral using `HAL_ADCEx_InjectedStart()`
- Wait for end of conversion using `HAL_ADC_PollForConversion()`, at this stage user can specify the value of timeout according to his end application
- To read the ADC converted values, use the `HAL_ADCEx_InjectedGetValue()` function.
- Stop the ADC peripheral using `HAL_ADCEx_InjectedStop()`

### **Interrupt mode IO operation**

- Start the ADC peripheral using `HAL_ADCEx_InjectedStart_IT()`
- Use `HAL_ADC_IRQHandler()` called under `ADC_IRQHandler()` Interrupt subroutine
- At ADC end of conversion `HAL_ADCEx_InjectedConvCpltCallback()` function is executed and user can add his own code by customization of function pointer `HAL_ADCEx_InjectedConvCpltCallback`
- In case of ADC Error, `HAL_ADCEx_InjectedErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_ADCEx_InjectedErrorCallback`
- Stop the ADC peripheral using `HAL_ADCEx_InjectedStop_IT()`

### **DMA mode IO operation**

- Start the ADC peripheral using `HAL_ADCEx_InjectedStart_DMA()`, at this stage the user specify the length of data to be transferred at each end of conversion
- At The end of data transfer ba `HAL_ADCEx_InjectedConvCpltCallback()` function is executed and user can add his own code by customization of function pointer `HAL_ADCEx_InjectedConvCpltCallback`
- In case of transfer Error, `HAL_ADCEx_InjectedErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_ADCEx_InjectedErrorCallback`
- Stop the ADC peripheral using `HAL_ADCEx_InjectedStop_DMA()`

### **Multi mode ADCs Regular channels configuration**

- Select the Multi mode ADC regular channels features (dual or triple mode) and configure the DMA mode using `HAL_ADCEx_MultiModeConfigChannel()` functions.
- Start the ADC peripheral using `HAL_ADCEx_MultiModeStart_DMA()`, at this stage the user specify the length of data to be transferred at each end of conversion

- Read the ADCs converted values using the HAL\_ADCEx\_MultiModeGetValue() function.

## 5.2.2 Extended features functions

This section provides functions allowing to:

- Start conversion of injected channel.
- Stop conversion of injected channel.
- Start multimode and enable DMA transfer.
- Stop multimode and disable DMA transfer.
- Get result of injected channel conversion.
- Get result of multimode conversion.
- Configure injected channels.
- Configure multimode.
- [\*\*HAL\\_ADCEx\\_InjectedStart\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_InjectedStart\\_IT\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_InjectedStop\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_InjectedPollForConversion\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_InjectedStop\\_IT\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_InjectedGetValue\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_MultiModeStart\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_MultiModeStop\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_MultiModeGetValue\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_InjectedConvCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_InjectedConfigChannel\(\)\*\*](#)
- [\*\*HAL\\_ADCEx\\_MultiModeConfigChannel\(\)\*\*](#)

## 5.2.3 Extended features functions

### 5.2.3.1 HAL\_ADCEx\_InjectedStart

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStart (</b> <b><i>ADC_HandleTypeDef * hadc</i></b> )
Function Description	Enables the selected ADC software start conversion of the injected channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 5.2.3.2 HAL\_ADCEx\_InjectedStart\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (</b> <b><i>ADC_HandleTypeDef * hadc</i></b> )
Function Description	Enables the interrupt and starts ADC conversion of injected channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 5.2.3.3 HAL\_ADCEx\_InjectedStop

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStop (</b> <b><i>ADC_HandleTypeDef * hadc</i></b> )
Function Description	Disables ADC and stop conversion of injected channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Caution: This function will stop also regular channels.</li> </ul>

### 5.2.3.4 HAL\_ADCEx\_InjectedPollForConversion

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (</b> <b><i>ADC_HandleTypeDef * hadc, uint32_t Timeout</i></b> )
Function Description	Poll for injected conversion complete.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> <li>• <b>Timeout</b> : Timeout value in millisecond.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 5.2.3.5 HAL\_ADCEx\_InjectedStop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (</b> <b><i>ADC_HandleTypeDef* hadc</i></b> )
Function Description	Disables the interrupt and stop ADC conversion of injected channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Caution: This function will stop also regular channels.</li> </ul>

### 5.2.3.6 HAL\_ADCEx\_InjectedGetValue

Function Name	<b>uint32_t HAL_ADCEx_InjectedGetValue (</b> <b><i>ADC_HandleTypeDef* hadc, uint32_t InjectedRank</i></b> )
Function Description	Gets the converted value from data register of injected channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> <li>• <b>InjectedRank</b> : the ADC injected rank. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>ADC_INJECTED_RANK_1</b> : Injected Channel1 selected</li> <li>– <b>ADC_INJECTED_RANK_2</b> : Injected Channel2 selected</li> <li>– <b>ADC_INJECTED_RANK_3</b> : Injected Channel3 selected</li> <li>– <b>ADC_INJECTED_RANK_4</b> : Injected Channel4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 5.2.3.7 HAL\_ADCEx\_MultiModeStart\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (</b> <b><i>ADC_HandleTypeDef</i> * hadc, uint32_t * pData, uint32_t Length)</b>
Function Description	Enables ADC DMA request after last transfer (Multi-ADC mode) and enables ADC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> <li>• <b>pData</b> : Pointer to buffer in which transferred from ADC peripheral to memory will be stored.</li> <li>• <b>Length</b> : The length of data to be transferred from ADC peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>Caution: This function must be used only with the ADC master.</li> </ul>

### 5.2.3.8 HAL\_ADCEx\_MultiModeStop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (</b> <b><i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Disables ADC DMA (multi-ADC mode) and disables ADC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 5.2.3.9 HAL\_ADCEx\_MultiModeGetValue

Function Name	<b>uint32_t HAL_ADCEx_MultiModeGetValue (</b> <b><i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Returns the last ADC1, ADC2 and ADC3 regular conversions results data in the selected multi mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>The converted data value.</b></li> </ul>

## Notes

- None.

**5.2.3.10 HAL\_ADCEx\_InjectedConvCpltCallback**

Function Name	<b>void HAL_ADCEx_InjectedConvCpltCallback (</b> <b><i>ADC_HandleTypeDef</i> * hadc)</b>
Function Description	Injected conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a <i>ADC_HandleTypeDef</i> structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**5.2.3.11 HAL\_ADCEx\_InjectedConfigChannel**

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (</b> <b><i>ADC_HandleTypeDef</i> * hadc, <i>ADC_InjectionConfTypeDef</i> * sConfigInjected)</b>
Function Description	Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a <i>ADC_HandleTypeDef</i> structure that contains the configuration information for the specified ADC.</li> <li>• <b>sConfigInjected</b> : ADC configuration structure for injected channel.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**5.2.3.12 HAL\_ADCEx\_MultiModeConfigChannel**

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel (</b> <b><i>ADC_HandleTypeDef</i> * hadc, <i>ADC_MultiModeTypeDef</i> *</b>
---------------	--

---

	<b>multimode)</b>
Function Description	Configures the ADC multi-mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc</b> : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> <li>• <b>multimode</b> : pointer to an ADC_MultiModeTypeDef structure that contains the configuration information for multimode.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 5.3 ADCEx Firmware driver defines

### 5.3.1 ADCEx

ADCEx

#### *ADCEx\_Common\_mode*

- #define: **ADC\_MODE\_INDEPENDENT** ((uint32\_t)0x00000000)
- #define: **ADC\_DUALMODE\_REGSIMULT\_INJECSIMULT** ((uint32\_t)ADC\_CCR\_MULTI\_0)
- #define: **ADC\_DUALMODE\_REGSIMULT.AlterTrig** ((uint32\_t)ADC\_CCR\_MULTI\_1)
- #define: **ADC\_DUALMODE\_INJECSIMULT** ((uint32\_t)(ADC\_CCR\_MULTI\_2 | ADC\_CCR\_MULTI\_0))
- #define: **ADC\_DUALMODE\_REGSIMULT** ((uint32\_t)(ADC\_CCR\_MULTI\_2 | ADC\_CCR\_MULTI\_1))
- #define: **ADC\_DUALMODE\_INTERL** ((uint32\_t)(ADC\_CCR\_MULTI\_2 | ADC\_CCR\_MULTI\_1 | ADC\_CCR\_MULTI\_0))

- #define: ***ADC\_DUALMODE\_ALTERTRIG*** ((*uint32\_t*)(***ADC\_CCR\_MULTI\_3 | ADC\_CCR\_MULTI\_0***))
- #define: ***ADC\_TRIPLEMODE\_REGSIMULT\_INJECSIMULT*** ((*uint32\_t*)(***ADC\_CCR\_MULTI\_4 | ADC\_CCR\_MULTI\_0***))
- #define: ***ADC\_TRIPLEMODE\_REGSIMULT\_AlterTrig*** ((*uint32\_t*)(***ADC\_CCR\_MULTI\_4 | ADC\_CCR\_MULTI\_1***))
- #define: ***ADC\_TRIPLEMODE\_INJECSIMULT*** ((*uint32\_t*)(***ADC\_CCR\_MULTI\_4 | ADC\_CCR\_MULTI\_2 | ADC\_CCR\_MULTI\_0***))
- #define: ***ADC\_TRIPLEMODE\_REGSIMULT*** ((*uint32\_t*)(***ADC\_CCR\_MULTI\_4 | ADC\_CCR\_MULTI\_2 | ADC\_CCR\_MULTI\_1***))
- #define: ***ADC\_TRIPLEMODE\_INTERL*** ((*uint32\_t*)(***ADC\_CCR\_MULTI\_4 | ADC\_CCR\_MULTI\_2 | ADC\_CCR\_MULTI\_1 | ADC\_CCR\_MULTI\_0***))
- #define: ***ADC\_TRIPLEMODE\_ALTERTRIG*** ((*uint32\_t*)(***ADC\_CCR\_MULTI\_4 | ADC\_CCR\_MULTI\_3 | ADC\_CCR\_MULTI\_0***))

#### ***ADCEx\_delay\_between\_2\_sampling\_phases***

- #define: ***ADC\_TWOSAMPLINGDELAY\_5CYCLES*** ((*uint32\_t*)0x00000000)
- #define: ***ADC\_TWOSAMPLINGDELAY\_6CYCLES*** ((*uint32\_t*)***ADC\_CCR\_DELAY\_0***)
- #define: ***ADC\_TWOSAMPLINGDELAY\_7CYCLES*** ((*uint32\_t*)***ADC\_CCR\_DELAY\_1***)

- #define: ***ADC\_TWOSAMPLINGDELAY\_8CYCLES***  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_1* | *ADC\_CCR\_DELAY\_0*))
- #define: ***ADC\_TWOSAMPLINGDELAY\_9CYCLES***  
((*uint32\_t*)*ADC\_CCR\_DELAY\_2*)
- #define: ***ADC\_TWOSAMPLINGDELAY\_10CYCLES***  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_2* | *ADC\_CCR\_DELAY\_0*))
- #define: ***ADC\_TWOSAMPLINGDELAY\_11CYCLES***  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_2* | *ADC\_CCR\_DELAY\_1*))
- #define: ***ADC\_TWOSAMPLINGDELAY\_12CYCLES***  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_2* | *ADC\_CCR\_DELAY\_1* | *ADC\_CCR\_DELAY\_0*))
- #define: ***ADC\_TWOSAMPLINGDELAY\_13CYCLES***  
((*uint32\_t*)*ADC\_CCR\_DELAY\_3*)
- #define: ***ADC\_TWOSAMPLINGDELAY\_14CYCLES***  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_3* | *ADC\_CCR\_DELAY\_0*))
- #define: ***ADC\_TWOSAMPLINGDELAY\_15CYCLES***  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_3* | *ADC\_CCR\_DELAY\_1*))
- #define: ***ADC\_TWOSAMPLINGDELAY\_16CYCLES***  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_3* | *ADC\_CCR\_DELAY\_1* | *ADC\_CCR\_DELAY\_0*))
- #define: ***ADC\_TWOSAMPLINGDELAY\_17CYCLES***  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_3* | *ADC\_CCR\_DELAY\_2*))

- #define: **ADC\_TWOSAMPLINGDELAY\_18CYCLES**  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_3* | *ADC\_CCR\_DELAY\_2* | *ADC\_CCR\_DELAY\_0*))
- #define: **ADC\_TWOSAMPLINGDELAY\_19CYCLES**  
((*uint32\_t*)(*ADC\_CCR\_DELAY\_3* | *ADC\_CCR\_DELAY\_2* | *ADC\_CCR\_DELAY\_1*))
- #define: **ADC\_TWOSAMPLINGDELAY\_20CYCLES** (*uint32\_t*)*ADC\_CCR\_DELAY*)

#### ***ADCEx\_Direct\_memory\_access\_mode\_for\_multi\_mode***

- #define: **ADC\_DMAACCESSMODE\_DISABLED** (*uint32\_t*)0x00000000)  
*DMA mode disabled*

- #define: **ADC\_DMAACCESSMODE\_1** (*uint32\_t*)*ADC\_CCR\_DMA\_0*)  
*DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)*

- #define: **ADC\_DMAACCESSMODE\_2** (*uint32\_t*)*ADC\_CCR\_DMA\_1*)  
*DMA mode 2 enabled (2 / 3 half-words by pairs - 2&1 then 1&3 then 3&2)*

- #define: **ADC\_DMAACCESSMODE\_3** (*uint32\_t*)*ADC\_CCR\_DMA*)  
*DMA mode 3 enabled (2 / 3 bytes by pairs - 2&1 then 1&3 then 3&2)*

#### ***ADCEx\_External\_trigger\_edge\_Injected***

- #define: **ADC\_EXTERNALTRIGINJECCONVEDGE\_NONE** (*uint32\_t*)0x00000000)
- #define: **ADC\_EXTERNALTRIGINJECCONVEDGE\_RISING**  
((*uint32\_t*)*ADC\_CR2\_JEXTEN\_0*)
- #define: **ADC\_EXTERNALTRIGINJECCONVEDGE\_FALLING**  
((*uint32\_t*)*ADC\_CR2\_JEXTEN\_1*)

- #define: **ADC\_EXTERNALTRIGINJECCONVEDGE\_RISINGFALLING**  
((uint32\_t)ADC\_CR2\_JEXTEN)

***ADCEx\_External\_trigger\_Source\_Injected***

- #define: **ADC\_EXTERNALTRIGINJECCONV\_T1\_CC4** ((uint32\_t)0x00000000)
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T1\_TRGO**  
((uint32\_t)ADC\_CR2\_JEXTSEL\_0)
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T2\_CC1**  
((uint32\_t)ADC\_CR2\_JEXTSEL\_1)
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T2\_TRGO**  
((uint32\_t)(ADC\_CR2\_JEXTSEL\_1 | ADC\_CR2\_JEXTSEL\_0))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T3\_CC2**  
((uint32\_t)ADC\_CR2\_JEXTSEL\_2)
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T3\_CC4**  
((uint32\_t)(ADC\_CR2\_JEXTSEL\_2 | ADC\_CR2\_JEXTSEL\_0))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T4\_CC1**  
((uint32\_t)(ADC\_CR2\_JEXTSEL\_2 | ADC\_CR2\_JEXTSEL\_1))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T4\_CC2**  
((uint32\_t)(ADC\_CR2\_JEXTSEL\_2 | ADC\_CR2\_JEXTSEL\_1 |  
ADC\_CR2\_JEXTSEL\_0))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T4\_CC3**  
((uint32\_t)ADC\_CR2\_JEXTSEL\_3)

- #define: **ADC\_EXTERNALTRIGINJECCONV\_T4\_TRGO**  
((*uint32\_t*)(ADC\_CR2\_JEXTSEL\_3 | ADC\_CR2\_JEXTSEL\_0))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T5\_CC4**  
((*uint32\_t*)(ADC\_CR2\_JEXTSEL\_3 | ADC\_CR2\_JEXTSEL\_1))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T5\_TRGO**  
((*uint32\_t*)(ADC\_CR2\_JEXTSEL\_3 | ADC\_CR2\_JEXTSEL\_1 |  
ADC\_CR2\_JEXTSEL\_0))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T8\_CC2**  
((*uint32\_t*)(ADC\_CR2\_JEXTSEL\_3 | ADC\_CR2\_JEXTSEL\_2))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T8\_CC3**  
((*uint32\_t*)(ADC\_CR2\_JEXTSEL\_3 | ADC\_CR2\_JEXTSEL\_2 |  
ADC\_CR2\_JEXTSEL\_0))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_T8\_CC4**  
((*uint32\_t*)(ADC\_CR2\_JEXTSEL\_3 | ADC\_CR2\_JEXTSEL\_2 |  
ADC\_CR2\_JEXTSEL\_1))
- #define: **ADC\_EXTERNALTRIGINJECCONV\_EXT\_IT15**  
((*uint32\_t*)ADC\_CR2\_JEXTSEL)

#### ***ADCEx\_injected\_channel\_selection***

- #define: **ADC\_INJECTED\_RANK\_1** ((*uint32\_t*)0x00000001)
- #define: **ADC\_INJECTED\_RANK\_2** ((*uint32\_t*)0x00000002)
- #define: **ADC\_INJECTED\_RANK\_3** ((*uint32\_t*)0x00000003)

- #define: ***ADC\_INJECTED\_RANK\_4 ((uint32\_t)0x00000004)***

## 6 HAL CAN Generic Driver

### 6.1 CAN Firmware driver registers structures

#### 6.1.1 CAN\_HandleTypeDef

*CAN\_HandleTypeDef* is defined in the `stm32f4xx_hal_can.h`

##### Data Fields

- *CAN\_TypeDef \* Instance*
- *CAN\_InitTypeDef Init*
- *CanTxMsgTypeDef \* pTxMsg*
- *CanRxMsgTypeDef \* pRxMsg*
- *\_\_IO HAL\_CAN\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- ***CAN\_TypeDef\* CAN\_HandleTypeDef::Instance***
  - Register base address
- ***CAN\_InitTypeDef CAN\_HandleTypeDef::Init***
  - CAN required parameters
- ***CanTxMsgTypeDef\* CAN\_HandleTypeDef::pTxMsg***
  - Pointer to transmit structure
- ***CanRxMsgTypeDef\* CAN\_HandleTypeDef::pRxMsg***
  - Pointer to reception structure
- ***\_\_IO HAL\_CAN\_StateTypeDef CAN\_HandleTypeDef::State***
  - CAN communication state
- ***HAL\_LockTypeDef CAN\_HandleTypeDef::Lock***
  - CAN locking object
- ***\_\_IO uint32\_t CAN\_HandleTypeDef::ErrorCode***
  - CAN Error code

#### 6.1.2 CAN\_InitTypeDef

*CAN\_InitTypeDef* is defined in the `stm32f4xx_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

### 6.1.3 CAN\_FilterConfTypeDef

*CAN\_FilterConfTypeDef* is defined in the `stm32f4xx_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\_FilterTypeDef::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\_FilterTypeDef::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\_FilterTypeDef::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\_FilterTypeDef::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\_FilterTypeDef::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\_FilterTypeDef::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\_FilterTypeDef::FilterMode*
  - Specifies the filter mode to be initialized. This parameter can be a value of *CAN\_filter\_mode*
- *uint32\_t CAN\_FilterTypeDef::FilterScale*
  - Specifies the filter scale. This parameter can be a value of *CAN\_filter\_scale*
- *uint32\_t CAN\_FilterTypeDef::FilterActivation*
  - Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- *uint32\_t CAN\_FilterTypeDef::BankNumber*
  - Select the start slave bank filter. This parameter must be a number between Min\_Data = 0 and Max\_Data = 28

### 6.1.4 CAN\_FIFOMailBox\_TypeDef

*CAN\_FIFOMailBox\_TypeDef* is defined in the stm32f439xx.h

**Data Fields**

- `__IO uint32_t RIR`
- `__IO uint32_t RDTR`
- `__IO uint32_t RDLR`
- `__IO uint32_t RDHR`

**Field Documentation**

- `__IO uint32_t CAN_FIFOMailBox_TypeDef::RIR`
  - CAN receive FIFO mailbox identifier register
- `__IO uint32_t CAN_FIFOMailBox_TypeDef::RDTR`
  - CAN receive FIFO mailbox data length control and time stamp register
- `__IO uint32_t CAN_FIFOMailBox_TypeDef::RDLR`
  - CAN receive FIFO mailbox data low register
- `__IO uint32_t CAN_FIFOMailBox_TypeDef::RDHR`
  - CAN receive FIFO mailbox data high register

### 6.1.5 CAN\_FilterRegister\_TypeDef

`CAN_FilterRegister_TypeDef` is defined in the `stm32f439xx.h`

**Data Fields**

- `__IO uint32_t FR1`
- `__IO uint32_t FR2`

**Field Documentation**

- `__IO uint32_t CAN_FilterRegister_TypeDef::FR1`
  - CAN Filter bank register 1
- `__IO uint32_t CAN_FilterRegister_TypeDef::FR2`
  - CAN Filter bank register 1

### 6.1.6 CAN\_TxMailBox\_TypeDef

`CAN_TxMailBox_TypeDef` is defined in the `stm32f439xx.h`

**Data Fields**

- `__IO uint32_t TIR`
- `__IO uint32_t TDTR`
- `__IO uint32_t TDLR`
- `__IO uint32_t TDHR`

**Field Documentation**

- `__IO uint32_t CAN_TxMailBox_TypeDef::TIR`
  - CAN TX mailbox identifier register
- `__IO uint32_t CAN_TxMailBox_TypeDef::TDTR`
  - CAN mailbox data length control and time stamp register
- `__IO uint32_t CAN_TxMailBox_TypeDef::TDLR`
  - CAN mailbox data low register
- `__IO uint32_t CAN_TxMailBox_TypeDef::TDHR`
  - CAN mailbox data high register

**6.1.7 CAN\_TypeDef**

`CAN_TypeDef` is defined in the `stm32f439xx.h`

**Data Fields**

- `__IO uint32_t MCR`
- `__IO uint32_t MSR`
- `__IO uint32_t TSR`
- `__IO uint32_t RF0R`
- `__IO uint32_t RF1R`
- `__IO uint32_t IER`
- `__IO uint32_t ESR`
- `__IO uint32_t BTR`
- `uint32_t RESERVED0`
- `CAN_TxMailBox_TypeDef sTxMailBox`
- `CAN_FIFOMailBox_TypeDef sFIFOMailBox`
- `uint32_t RESERVED1`
- `__IO uint32_t FMR`
- `__IO uint32_t FM1R`
- `uint32_t RESERVED2`
- `__IO uint32_t FS1R`
- `uint32_t RESERVED3`
- `__IO uint32_t FFA1R`
- `uint32_t RESERVED4`
- `__IO uint32_t FA1R`
- `uint32_t RESERVED5`
- `CAN_FilterRegister_TypeDef sFilterRegister`

**Field Documentation**

- `__IO uint32_t CAN_TypeDef::MCR`
  - CAN master control register, Address offset: 0x00
- `__IO uint32_t CAN_TypeDef::MSR`
  - CAN master status register, Address offset: 0x04
- `__IO uint32_t CAN_TypeDef::TSR`
  - CAN transmit status register, Address offset: 0x08
- `__IO uint32_t CAN_TypeDef::RF0R`

- CAN receive FIFO 0 register, Address offset: 0x0C
- **`__IO uint32_t CAN_TypeDef::RF1R`**
  - CAN receive FIFO 1 register, Address offset: 0x10
- **`__IO uint32_t CAN_TypeDef::IER`**
  - CAN interrupt enable register, Address offset: 0x14
- **`__IO uint32_t CAN_TypeDef::ESR`**
  - CAN error status register, Address offset: 0x18
- **`__IO uint32_t CAN_TypeDef::BTR`**
  - CAN bit timing register, Address offset: 0x1C
- **`uint32_t CAN_TypeDef::RESERVED0[88]`**
  - Reserved, 0x020 - 0x17F
- **`CAN_TxMailBox_TypeDef CAN_TypeDef::sTxMailBox[3]`**
  - CAN Tx MailBox, Address offset: 0x180 - 0x1AC
- **`CAN_FIFOMailBox_TypeDef CAN_TypeDef::sFIFOMailBox[2]`**
  - CAN FIFO MailBox, Address offset: 0x1B0 - 0x1CC
- **`uint32_t CAN_TypeDef::RESERVED1[12]`**
  - Reserved, 0x1D0 - 0x1FF
- **`__IO uint32_t CAN_TypeDef::FMR`**
  - CAN filter master register, Address offset: 0x200
- **`__IO uint32_t CAN_TypeDef::FM1R`**
  - CAN filter mode register, Address offset: 0x204
- **`uint32_t CAN_TypeDef::RESERVED2`**
  - Reserved, 0x208
- **`__IO uint32_t CAN_TypeDef::FS1R`**
  - CAN filter scale register, Address offset: 0x20C
- **`uint32_t CAN_TypeDef::RESERVED3`**
  - Reserved, 0x210
- **`__IO uint32_t CAN_TypeDef::FFA1R`**
  - CAN filter FIFO assignment register, Address offset: 0x214
- **`uint32_t CAN_TypeDef::RESERVED4`**
  - Reserved, 0x218
- **`__IO uint32_t CAN_TypeDef::FA1R`**
  - CAN filter activation register, Address offset: 0x21C
- **`uint32_t CAN_TypeDef::RESERVED5[8]`**
  - Reserved, 0x220-0x23F
- **`CAN_FilterRegister_TypeDef CAN_TypeDef::sFilterRegister[28]`**
  - CAN Filter Register, Address offset: 0x240-0x31C

## 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 `__CAN1_CLK_ENABLE()` for CAN1 and `__CAN2_CLK_ENABLE()` for CAN2. In case you are using CAN2 only, you have to enable the CAN1 clock.
2. CAN pins configuration

- Enable the clock for the CAN GPIOs using the following function:  
`_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 `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\_DelInit\(\)`](#)
- [`HAL\_CAN\_MspInit\(\)`](#)
- [`HAL\_CAN\_MspDelInit\(\)`](#)

### 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 Initialization and de-initialization functions

#### 6.2.5.1 HAL\_CAN\_Init

Function Name	<code>HAL_StatusTypeDef HAL_CAN_Init ( CAN_HandleTypeDef * hcan)</code>
Function Description	Initializes the CAN peripheral according to the specified parameters in the CAN_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 6.2.5.2 HAL\_CAN\_ConfigFilter

Function Name	<b>HAL_StatusTypeDef HAL_CAN_ConfigFilter ( <i>CAN_HandleTypeDef</i> * hcan, <i>CAN_FilterConfTypeDef</i> * sFilterConfig)</b>
Function Description	Configures the CAN reception filter according to the specified parameters in the CAN_FilterInitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> <li>• <b>sFilterConfig</b> : pointer to a CAN_FilterConfTypeDef structure that contains the filter configuration information.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 6.2.5.3 HAL\_CAN\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_CAN_DeInit ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Deinitializes the CANx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 6.2.5.4 HAL\_CAN\_MspInit

Function Name	<b>void HAL_CAN_MspInit ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Initializes the CAN MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## Notes

- None.

### 6.2.5.5 HAL\_CAN\_MspDeInit

Function Name	<b>void HAL_CAN_MspDeInit ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Deinitializes the CAN MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 6.2.6 IO operation functions

### 6.2.6.1 HAL\_CAN\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Transmit ( <i>CAN_HandleTypeDef</i> * hcan, uint32_t Timeout)</b>
Function Description	Initiates and transmits a CAN frame message.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li><li>• <b>Timeout</b> : Specify Timeout value</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.2 HAL\_CAN\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Transmit_IT ( <i>CAN_HandleTypeDef</i> * hcan)</b>
---------------	---

Function Description	Initiates and transmits a CAN frame message.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.3 HAL\_CAN\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Receive (</b> <b>CAN_HandleTypeDef * hcan, uint8_t FIFONumber, uint32_t Timeout)</b>
Function Description	Receives a correct CAN frame.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li><li>• <b>FIFONumber</b> : FIFO Number value</li><li>• <b>Timeout</b> : Specify Timeout value</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.4 HAL\_CAN\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Receive_IT (</b> <b>CAN_HandleTypeDef * hcan, uint8_t FIFONumber)</b>
Function Description	Receives a correct CAN frame.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : Pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li><li>• <b>FIFONumber</b> : Specify the FIFO number</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.5 HAL\_CAN\_Sleep

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Sleep ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Enters the Sleep (low power) mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.6 HAL\_CAN\_WakeUp

Function Name	<b>HAL_StatusTypeDef HAL_CAN_WakeUp ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Wakes up the CAN peripheral from sleep mode, after that the CAN peripheral is in the normal mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.7 HAL\_CAN\_IRQHandler

Function Name	<b>void HAL_CAN_IRQHandler ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Handles CAN interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.8 HAL\_CAN\_TxCpltCallback

Function Name	<b>void HAL_CAN_TxCpltCallback ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Transmission complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.9 HAL\_CAN\_RxCpltCallback

Function Name	<b>void HAL_CAN_RxCpltCallback ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Transmission complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.6.10 HAL\_CAN\_ErrorCallback

Function Name	<b>void HAL_CAN_ErrorCallback ( <i>CAN_HandleTypeDef</i> * hcan)</b>
Function Description	Error CAN callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 6.2.7 Peripheral State and Error functions

### 6.2.7.1 HAL\_CAN\_GetState

Function Name	<code>HAL_CAN_StateTypeDef HAL_CAN_GetState (CAN_HandleTypeDef * hcan)</code>
Function Description	return the CAN state
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 6.2.7.2 HAL\_CAN\_GetError

Function Name	<code>uint32_t HAL_CAN_GetError (CAN_HandleTypeDef * hcan)</code>
Function Description	Return the CAN error code.
Parameters	<ul style="list-style-type: none"><li>• <b>hcan</b> : pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>CAN Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 6.3 CAN Firmware driver defines

### 6.3.1 CAN

CAN

*CAN\_Exported\_Constants*

- #define: *INAK\_TIMEOUT ((uint32\_t)0x0000FFFF)*

- #define: **SLAK\_TIMEOUT** ((*uint32\_t*)0x0000FFFF)
  
- #define: **CAN\_TXMAILBOX\_0** ((*uint8\_t*)0x00)
  
- #define: **CAN\_TXMAILBOX\_1** ((*uint8\_t*)0x01)
  
- #define: **CAN\_TXMAILBOX\_2** ((*uint8\_t*)0x02)

#### **CAN\_filter\_FIFO**

- #define: **CAN\_FILTER\_FIFO0** ((*uint8\_t*)0x00)  
*Filter FIFO 0 assignment for filter x*
  
- #define: **CAN\_FILTER\_FIFO1** ((*uint8\_t*)0x01)  
*Filter FIFO 1 assignment for filter x*
  
- #define: **CAN\_FilterFIFO0 CAN\_FILTER\_FIFO0**
  
  
- #define: **CAN\_FilterFIFO1 CAN\_FILTER\_FIFO1**

#### **CAN\_filter\_mode**

- #define: **CAN\_FILTERMODE\_IDMASK** ((*uint8\_t*)0x00)  
*Identifier mask mode*
  
- #define: **CAN\_FILTERMODE\_IDLIST** ((*uint8\_t*)0x01)  
*Identifier list mode*

#### **CAN\_filter\_scale**

- #define: **CAN\_FILTERSCALE\_16BIT** ((*uint8\_t*)0x00)  
*Two 16-bit filters*

- #define: **CAN\_FILTERSCALE\_32BIT** ((*uint8\_t*)0x01)

*One 32-bit filter*

#### **CAN\_flags**

- #define: **CAN\_FLAG\_RQCP0** ((*uint32\_t*)0x00000500)

*Request MailBox0 flag*

- #define: **CAN\_FLAG\_RQCP1** ((*uint32\_t*)0x00000508)

*Request MailBox1 flag*

- #define: **CAN\_FLAG\_RQCP2** ((*uint32\_t*)0x00000510)

*Request MailBox2 flag*

- #define: **CAN\_FLAG\_TXOK0** ((*uint32\_t*)0x00000501)

*Transmission OK MailBox0 flag*

- #define: **CAN\_FLAG\_TXOK1** ((*uint32\_t*)0x00000509)

*Transmission OK MailBox1 flag*

- #define: **CAN\_FLAG\_TXOK2** ((*uint32\_t*)0x00000511)

*Transmission OK MailBox2 flag*

- #define: **CAN\_FLAG\_TME0** ((*uint32\_t*)0x0000051A)

*Transmit mailbox 0 empty flag*

- #define: **CAN\_FLAG\_TME1** ((*uint32\_t*)0x0000051B)

*Transmit mailbox 0 empty flag*

- #define: **CAN\_FLAG\_TME2** ((*uint32\_t*)0x0000051C)

*Transmit mailbox 0 empty flag*

- #define: **CAN\_FLAG\_FF0** ((*uint32\_t*)0x00000203)

*FIFO 0 Full flag*

- #define: **CAN\_FLAG\_FOV0** ((*uint32\_t*)0x00000204)

*FIFO 0 Overrun flag*

- #define: **CAN\_FLAG\_FF1** ((*uint32\_t*)0x00000403)  
*FIFO 1 Full flag*
- #define: **CAN\_FLAG\_FOV1** ((*uint32\_t*)0x00000404)  
*FIFO 1 Overrun flag*
- #define: **CAN\_FLAG\_WKU** ((*uint32\_t*)0x00000103)  
*Wake up flag*
- #define: **CAN\_FLAG\_SLAK** ((*uint32\_t*)0x00000101)  
*Sleep acknowledge flag*
- #define: **CAN\_FLAG\_SLAKI** ((*uint32\_t*)0x00000104)  
*Sleep acknowledge flag*
- #define: **CAN\_FLAG\_EWG** ((*uint32\_t*)0x00000300)  
*Error warning flag*
- #define: **CAN\_FLAG\_EPV** ((*uint32\_t*)0x00000301)  
*Error passive flag*
- #define: **CAN\_FLAG\_BOF** ((*uint32\_t*)0x00000302)  
*Bus-Off flag*

***CAN\_identifier\_type***

- #define: **CAN\_ID\_STD** ((*uint32\_t*)0x00000000)  
*Standard Id*
- #define: **CAN\_ID\_EXT** ((*uint32\_t*)0x00000004)  
*Extended Id*

***CAN\_InitStatus***

- #define: **CAN\_INITSTATUS\_FAILED** ((*uint8\_t*)0x00)  
*CAN initialization failed*

- #define: **CAN\_INITSTATUS\_SUCCESS** ((uint8\_t)0x01)

*CAN initialization OK*

***CAN\_interrupts***

- #define: **CAN\_IT\_TME** ((uint32\_t)**CAN\_IER\_TMEIE**)

*Transmit mailbox empty interrupt*

- #define: **CAN\_IT\_FMP0** ((uint32\_t)**CAN\_IER\_FMPIE0**)

*FIFO 0 message pending interrupt*

- #define: **CAN\_IT\_FF0** ((uint32\_t)**CAN\_IER\_FFIE0**)

*FIFO 0 full interrupt*

- #define: **CAN\_IT\_FOV0** ((uint32\_t)**CAN\_IER\_FOVIE0**)

*FIFO 0 overrun interrupt*

- #define: **CAN\_IT\_FMP1** ((uint32\_t)**CAN\_IER\_FMPIE1**)

*FIFO 1 message pending interrupt*

- #define: **CAN\_IT\_FF1** ((uint32\_t)**CAN\_IER\_FFIE1**)

*FIFO 1 full interrupt*

- #define: **CAN\_IT\_FOV1** ((uint32\_t)**CAN\_IER\_FOVIE1**)

*FIFO 1 overrun interrupt*

- #define: **CAN\_IT\_WKU** ((uint32\_t)**CAN\_IER\_WKUIE**)

*Wake-up interrupt*

- #define: **CAN\_IT\_SLK** ((uint32\_t)**CAN\_IER\_SLKIE**)

*Sleep acknowledge interrupt*

- #define: **CAN\_IT\_EWG** ((uint32\_t)**CAN\_IER\_EWGIE**)

*Error warning interrupt*

- #define: **CAN\_IT\_EPV** ((uint32\_t)**CAN\_IER\_EPVIE**)

*Error passive interrupt*

- #define: **CAN\_IT\_BOF** ((*uint32\_t*)**CAN\_IER\_BOFIE**)  
*Bus-off interrupt*

- #define: **CAN\_IT\_LEC** ((*uint32\_t*)**CAN\_IER\_LECIE**)  
*Last error code interrupt*

- #define: **CAN\_IT\_ERR** ((*uint32\_t*)**CAN\_IER\_ERRIE**)  
*Error Interrupt*

- #define: **CAN\_IT\_RQCP0 CAN\_IT\_TME**

- #define: **CAN\_IT\_RQCP1 CAN\_IT\_TME**

- #define: **CAN\_IT\_RQCP2 CAN\_IT\_TME**

#### **CAN\_operating\_mode**

- #define: **CAN\_MODE\_NORMAL** ((*uint32\_t*)0x00000000)  
*Normal mode*

- #define: **CAN\_MODE\_LOOPBACK** ((*uint32\_t*)**CAN\_BTR\_LBKM**)  
*Loopback mode*

- #define: **CAN\_MODE\_SILENT** ((*uint32\_t*)**CAN\_BTR\_SILM**)  
*Silent mode*

- #define: **CAN\_MODE\_SILENT\_LOOPBACK** ((*uint32\_t*)(**CAN\_BTR\_LBKM** | **CAN\_BTR\_SILM**))

*Loopback combined with silent mode*

#### **CAN\_receive\_FIFO\_number\_constants**

- #define: **CAN\_FIFO0** ((*uint8\_t*)0x00)  
*CAN FIFO 0 used to receive*

- #define: **CAN\_FIFO1** ((*uint8\_t*)0x01)

*CAN FIFO 1 used to receive*

***CAN\_remote\_transmission\_request***

- #define: **CAN\_RTR\_DATA** ((*uint32\_t*)0x00000000)

*Data frame*

- #define: **CAN\_RTR\_REMOTE** ((*uint32\_t*)0x00000002)

*Remote frame*

***CAN\_synchronisation\_jump\_width***

- #define: **CAN\_SJW\_1TQ** ((*uint32\_t*)0x00000000)

*1 time quantum*

- #define: **CAN\_SJW\_2TQ** ((*uint32\_t*)**CAN\_BTR\_SJW\_0**)

*2 time quantum*

- #define: **CAN\_SJW\_3TQ** ((*uint32\_t*)**CAN\_BTR\_SJW\_1**)

*3 time quantum*

- #define: **CAN\_SJW\_4TQ** ((*uint32\_t*)**CAN\_BTR\_SJW**)

*4 time quantum*

***CAN\_time\_quantum\_in\_bit\_segment\_1***

- #define: **CAN\_BS1\_1TQ** ((*uint32\_t*)0x00000000)

*1 time quantum*

- #define: **CAN\_BS1\_2TQ** ((*uint32\_t*)**CAN\_BTR\_TS1\_0**)

*2 time quantum*

- #define: **CAN\_BS1\_3TQ** ((*uint32\_t*)**CAN\_BTR\_TS1\_1**)

*3 time quantum*

- #define: **CAN\_BS1\_4TQ** ((*uint32\_t*)(**CAN\_BTR\_TS1\_1** | **CAN\_BTR\_TS1\_0**))

*4 time quantum*

- #define: **CAN\_BS1\_5TQ ((uint32\_t)CAN\_BTR\_TS1\_2)**  
*5 time quantum*
- #define: **CAN\_BS1\_6TQ ((uint32\_t)(CAN\_BTR\_TS1\_2 | CAN\_BTR\_TS1\_0))**  
*6 time quantum*
- #define: **CAN\_BS1\_7TQ ((uint32\_t)(CAN\_BTR\_TS1\_2 | CAN\_BTR\_TS1\_1))**  
*7 time quantum*
- #define: **CAN\_BS1\_8TQ ((uint32\_t)(CAN\_BTR\_TS1\_2 | CAN\_BTR\_TS1\_1 | CAN\_BTR\_TS1\_0))**  
*8 time quantum*
- #define: **CAN\_BS1\_9TQ ((uint32\_t)CAN\_BTR\_TS1\_3)**  
*9 time quantum*
- #define: **CAN\_BS1\_10TQ ((uint32\_t)(CAN\_BTR\_TS1\_3 | CAN\_BTR\_TS1\_0))**  
*10 time quantum*
- #define: **CAN\_BS1\_11TQ ((uint32\_t)(CAN\_BTR\_TS1\_3 | CAN\_BTR\_TS1\_1))**  
*11 time quantum*
- #define: **CAN\_BS1\_12TQ ((uint32\_t)(CAN\_BTR\_TS1\_3 | CAN\_BTR\_TS1\_1 | CAN\_BTR\_TS1\_0))**  
*12 time quantum*
- #define: **CAN\_BS1\_13TQ ((uint32\_t)(CAN\_BTR\_TS1\_3 | CAN\_BTR\_TS1\_2))**  
*13 time quantum*
- #define: **CAN\_BS1\_14TQ ((uint32\_t)(CAN\_BTR\_TS1\_3 | CAN\_BTR\_TS1\_2 | CAN\_BTR\_TS1\_0))**  
*14 time quantum*
- #define: **CAN\_BS1\_15TQ ((uint32\_t)(CAN\_BTR\_TS1\_3 | CAN\_BTR\_TS1\_2 | CAN\_BTR\_TS1\_1))**  
*15 time quantum*
- #define: **CAN\_BS1\_16TQ ((uint32\_t)CAN\_BTR\_TS1)**

*16 time quantum*

***CAN\_time\_quantum\_in\_bit\_segment\_2***

- #define: ***CAN\_BS2\_1TQ ((uint32\_t)0x00000000)***

*1 time quantum*

- #define: ***CAN\_BS2\_2TQ ((uint32\_t)CAN\_BTR\_TS2\_0)***

*2 time quantum*

- #define: ***CAN\_BS2\_3TQ ((uint32\_t)CAN\_BTR\_TS2\_1)***

*3 time quantum*

- #define: ***CAN\_BS2\_4TQ ((uint32\_t)(CAN\_BTR\_TS2\_1 | CAN\_BTR\_TS2\_0))***

*4 time quantum*

- #define: ***CAN\_BS2\_5TQ ((uint32\_t)CAN\_BTR\_TS2\_2)***

*5 time quantum*

- #define: ***CAN\_BS2\_6TQ ((uint32\_t)(CAN\_BTR\_TS2\_2 | CAN\_BTR\_TS2\_0))***

*6 time quantum*

- #define: ***CAN\_BS2\_7TQ ((uint32\_t)(CAN\_BTR\_TS2\_2 | CAN\_BTR\_TS2\_1))***

*7 time quantum*

- #define: ***CAN\_BS2\_8TQ ((uint32\_t)CAN\_BTR\_TS2)***

*8 time quantum*

***CAN\_transmit\_constants***

- #define: ***CAN\_TXSTATUS\_FAILED ((uint8\_t)0x00)***

*CAN transmission failed*

- #define: ***CAN\_TXSTATUS\_OK ((uint8\_t)0x01)***

*CAN transmission succeeded*

- #define: ***CAN\_TXSTATUS\_PENDING ((uint8\_t)0x02)***

*CAN transmission pending*

- #define: **CAN\_TXSTATUS\_NOMAILBOX** (*uint8\_t*)0x04  
*CAN cell did not provide CAN\_TxStatus\_NoMailBox*

## 7 HAL CORTEX Generic Driver

### 7.1 CORTEX Firmware driver API description

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

#### 7.1.1 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. Refer to STM32F3xx/STM32F4xx Cortex-M4 programming manual (PM0214) for details.
2. Configure the priority of the selected IRQ Channels using HAL\_NVIC\_SetPriority().
3. Enable the selected IRQ Channels using HAL\_NVIC\_EnableIRQ().
4. please refer to programing manual for details in how to configure priority. 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)

##### How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for 1 msec interrupts.

- 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 stm32f4xx\_hal\_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL\_NVIC\_SetPriority(SysTick\_IRQn,...) function just after the HAL\_SYSTICK\_Config() function call. The HAL\_NVIC\_SetPriority() call the NVIC\_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL\_SYSTICK\_Config() function
  - Reload Value should not exceed 0xFFFFFFF

## 7.1.2 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()`

## 7.1.3 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()`

## 7.1.4 Initialization and de-initialization functions

### 7.1.4.1 HAL\_NVIC\_SetPriorityGrouping

Function Name	<code>void HAL_NVIC_SetPriorityGrouping ( uint32_t PriorityGroup)</code>
Function Description	Sets the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>PriorityGroup</b> : The priority grouping bits length. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>NVIC_PRIORITYGROUP_0</code> : 0 bits for pre-emption priority 4 bits for subpriority</li> <li>– <code>NVIC_PRIORITYGROUP_1</code> : 1 bits for pre-emption priority 3 bits for subpriority</li> <li>– <code>NVIC_PRIORITYGROUP_2</code> : 2 bits for pre-emption priority 2 bits for subpriority</li> <li>– <code>NVIC_PRIORITYGROUP_3</code> : 3 bits for pre-emption priority 1 bits for subpriority</li> <li>– <code>NVIC_PRIORITYGROUP_4</code> : 4 bits for pre-emption priority 0 bits for subpriority</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption</li> </ul>

is no more possible. The pending IRQ priority will be managed only by the subpriority.

#### 7.1.4.2 HAL\_NVIC\_SetPriority

Function Name	<b>void HAL_NVIC_SetPriority ( IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)</b>
Function Description	Sets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn</b> : External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32f4xx.h file)</li> <li>• <b>PreemptPriority</b> : The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority</li> <li>• <b>SubPriority</b> : the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 7.1.4.3 HAL\_NVIC\_EnableIRQ

Function Name	<b>void HAL_NVIC_EnableIRQ ( IRQn_Type IRQn)</b>
Function Description	Enables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn</b> : External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32f4xx.h file)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.</li> </ul>

#### 7.1.4.4 HAL\_NVIC\_DisableIRQ

Function Name	<b>void HAL_NVIC_DisableIRQ ( IRQn_Type IRQn)</b>
Function Description	Disables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"><li><b>IRQn</b> : External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to <a href="#">stm32f4xx.h</a> file)</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

#### 7.1.4.5 HAL\_NVIC\_SystemReset

Function Name	<b>void HAL_NVIC_SystemReset ( void )</b>
Function Description	Initiates a system reset request to reset the MCU.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>

#### 7.1.4.6 HAL\_SYSTICK\_Config

Function Name	<b>uint32_t HAL_SYSTICK_Config ( uint32_t TicksNumb)</b>
Function Description	Initializes the System Timer and its interrupt, and starts the System Tick Timer.
Parameters	<ul style="list-style-type: none"><li><b>TicksNumb</b> : Specifies the ticks Number of ticks between two interrupts.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>status</b> : - 0 Function succeeded.<ul style="list-style-type: none"><li>- 1 Function failed.</li></ul></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 7.1.5 Peripheral Control functions

### 7.1.5.1 HAL\_NVIC\_GetPriorityGrouping

Function Name	<code>uint32_t HAL_NVIC_GetPriorityGrouping ( void )</code>
Function Description	Gets the priority grouping field from the NVIC Interrupt Controller.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Priority grouping field (SCB-&gt;AIRCR [10:8] PRIGROUP field)</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 7.1.5.2 HAL\_NVIC\_GetPriority

Function Name	<code>void HAL_NVIC_GetPriority ( IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)</code>
Function Description	Gets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn</b> : External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32f4xx.h file)</li> <li>• <b>PriorityGroup</b> : the priority grouping bits length. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>NVIC_PRIORITYGROUP_0</b> : 0 bits for pre-emption priority 4 bits for subpriority</li> <li>– <b>NVIC_PRIORITYGROUP_1</b> : 1 bits for pre-emption priority 3 bits for subpriority</li> <li>– <b>NVIC_PRIORITYGROUP_2</b> : 2 bits for pre-emption priority 2 bits for subpriority</li> <li>– <b>NVIC_PRIORITYGROUP_3</b> : 3 bits for pre-emption priority 1 bits for subpriority</li> <li>– <b>NVIC_PRIORITYGROUP_4</b> : 4 bits for pre-emption priority 0 bits for subpriority</li> </ul> </li> <li>• <b>pPreemptPriority</b> : Pointer on the Preemptive priority value (starting from 0).</li> <li>• <b>pSubPriority</b> : Pointer on the Subpriority value (starting from 0).</li> </ul>

---

Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 7.1.5.3 HAL\_NVIC\_SetPendingIRQ

Function Name	<b>void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)</b>
Function Description	Sets Pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>IRQn</b> : External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32f4xx.h file)</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 7.1.5.4 HAL\_NVIC\_GetPendingIRQ

Function Name	<b>uint32_t HAL_NVIC_GetPendingIRQ ( IRQn_Type IRQn)</b>
Function Description	Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).
Parameters	<ul style="list-style-type: none"><li>• <b>IRQn</b> : External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32f4xx.h file)</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>status</b> : - <b>0</b> <b>Interrupt status is not pending.</b> – <b>1</b> <b>Interrupt status is pending.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 7.1.5.5 HAL\_NVIC\_ClearPendingIRQ

---

Function Name	<b>void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)</b>
Function Description	Clears the pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn</b> : External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32f4xx.h file)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 7.1.5.6 HAL\_NVIC\_GetActive

Function Name	<b>uint32_t HAL_NVIC_GetActive ( IRQn_Type IRQn)</b>
Function Description	Gets active interrupt ( reads the active register in NVIC and returns the active bit).
Parameters	<ul style="list-style-type: none"> <li>• <b>IRQn</b> : External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to stm32f4xx.h file)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>status</b> : - <b>0</b> <b>Interrupt status is not pending.</b> <ul style="list-style-type: none"> <li>- <b>1</b> <b>Interrupt status is pending.</b></li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 7.1.5.7 HAL\_SYSTICK\_CLKSourceConfig

Function Name	<b>void HAL_SYSTICK_CLKSourceConfig ( uint32_t CLKSource)</b>
Function Description	Configures the SysTick clock source.
Parameters	<ul style="list-style-type: none"> <li>• <b>CLKSource</b> : specifies the SysTick clock source. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <b>SYSTICK_CLKSOURCE_HCLK_DIV8</b> : AHB clock divided by 8 selected as SysTick clock source.</li> <li>- <b>SYSTICK_CLKSOURCE_HCLK</b> : AHB clock selected as SysTick clock source.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 7.1.5.8 HAL\_SYSTICK\_IRQHandler

Function Name	<b>void HAL_SYSTICK_IRQHandler ( void )</b>
Function Description	This function handles SYSTICK interrupt request.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 7.1.5.9 HAL\_SYSTICK\_Callback

Function Name	<b>void HAL_SYSTICK_Callback ( void )</b>
Function Description	SYSTICK callback.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 7.2 CORTEX Firmware driver defines

### 7.2.1 CORTEX

CORTEX

*CORTEX\_Preemption\_Priority\_Group*

- #define: **NVIC\_PRIORITYGROUP\_0 ((uint32\_t)0x00000007)**  
*0 bits for pre-emption priority 4 bits for subpriority*

- #define: **NVIC\_PRIORITYGROUP\_1 ((uint32\_t)0x00000006)**  
*1 bits for pre-emption priority 3 bits for subpriority*

- #define: **NVIC\_PRIORITYGROUP\_2** ((*uint32\_t*)0x00000005)  
*2 bits for pre-emption priority 2 bits for subpriority*
  
  - #define: **NVIC\_PRIORITYGROUP\_3** ((*uint32\_t*)0x00000004)  
*3 bits for pre-emption priority 1 bits for subpriority*
  
  - #define: **NVIC\_PRIORITYGROUP\_4** ((*uint32\_t*)0x00000003)  
*4 bits for pre-emption priority 0 bits for subpriority*
- CORTEX\_SysTick\_clock\_source**
- #define: **SYSTICK\_CLKSOURCE\_HCLK\_DIV8** ((*uint32\_t*)0x00000000)
  
  - #define: **SYSTICK\_CLKSOURCE\_HCLK** ((*uint32\_t*)0x00000004)

## 8 HAL CRC Generic Driver

### 8.1 CRC Firmware driver registers structures

#### 8.1.1 CRC\_HandleTypeDef

*CRC\_HandleTypeDef* is defined in the `stm32f4xx_hal_crc.h`

##### Data Fields

- *CRC\_TypeDef \* Instance*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_CRC\_StateTypeDef State*

##### Field Documentation

- *CRC\_TypeDef\* CRC\_HandleTypeDef::Instance*
  - Register base address
- *HAL\_LockTypeDef CRC\_HandleTypeDef::Lock*
  - CRC locking object
- *\_\_IO HAL\_CRC\_StateTypeDef CRC\_HandleTypeDef::State*
  - CRC communication state

#### 8.1.2 CRC\_TypeDef

*CRC\_TypeDef* is defined in the `stm32f439xx.h`

##### Data Fields

- *\_\_IO uint32\_t DR*
- *\_\_IO uint8\_t IDR*
- *uint8\_t RESERVED0*
- *uint16\_t RESERVED1*
- *\_\_IO uint32\_t CR*

##### Field Documentation

- *\_\_IO uint32\_t CRC\_TypeDef::DR*
  - CRC Data register, Address offset: 0x00
- *\_\_IO uint8\_t CRC\_TypeDef::IDR*
  - CRC Independent data register, Address offset: 0x04
- *uint8\_t CRC\_TypeDef::RESERVED0*
  - Reserved, 0x05
- *uint16\_t CRC\_TypeDef::RESERVED1*
  - Reserved, 0x06
- *\_\_IO uint32\_t CRC\_TypeDef::CR*

- CRC Control register, Address offset: 0x08

## 8.2 CRC Firmware driver API description

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

### 8.2.1 How to use this driver

The CRC HAL driver can be used as follows:

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

### 8.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the `CRC_InitTypeDef` and create the associated handle
- Deinitialize the CRC peripheral
- Initialize the CRC MSP
- Deinitialize CRC MSP
- [`HAL\_CRC\_Init\(\)`](#)
- [`HAL\_CRC\_DelInit\(\)`](#)
- [`HAL\_CRC\_MspInit\(\)`](#)
- [`HAL\_CRC\_MspDelInit\(\)`](#)

### 8.2.3 Peripheral Control functions

This section provides functions allowing to:

- Compute the 32-bit CRC value of 32-bit data buffer, using combination of the previous CRC value and the new one.
- Compute the 32-bit CRC value of 32-bit data buffer, independently of the previous CRC value.
- [`HAL\_CRC\_Accumulate\(\)`](#)
- [`HAL\_CRC\_Calculate\(\)`](#)

### 8.2.4 Peripheral State functions

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

- [`HAL\_CRC\_GetState\(\)`](#)

## 8.2.5 Initialization and de-initialization functions

### 8.2.5.1 HAL\_CRC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CRC_Init ( <i>CRC_HandleTypeDef</i> * <i>hcrc</i>)</b>
Function Description	Initializes the CRC according to the specified parameters in the <i>CRC_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b> : pointer to a <i>CRC_HandleTypeDef</i> structure that contains the configuration information for CRC</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 8.2.5.2 HAL\_CRC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_CRC_DeInit ( <i>CRC_HandleTypeDef</i> * <i>hcrc</i>)</b>
Function Description	Deinitializes the CRC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b> : pointer to a <i>CRC_HandleTypeDef</i> structure that contains the configuration information for CRC</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 8.2.5.3 HAL\_CRC\_MspInit

Function Name	<b>void HAL_CRC_MspInit ( <i>CRC_HandleTypeDef</i> * <i>hcrc</i>)</b>
Function Description	Initializes the CRC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b> : pointer to a <i>CRC_HandleTypeDef</i> structure that contains the configuration information for CRC</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 8.2.5.4 HAL\_CRC\_MspDeInit

Function Name	<b>void HAL_CRC_MspDeInit ( <i>CRC_HandleTypeDef</i> * hcrc)</b>
Function Description	Deinitializes the CRC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b> : pointer to a <i>CRC_HandleTypeDef</i> structure that contains the configuration information for CRC</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 8.2.6 Peripheral Control functions

### 8.2.6.1 HAL\_CRC\_Accumulate

Function Name	<b>uint32_t HAL_CRC_Accumulate ( <i>CRC_HandleTypeDef</i> * hcrc,                                   uint32_t pBuffer, uint32_t BufferLength)</b>
Function Description	Computes the 32-bit CRC of 32-bit data buffer using combination of the previous CRC value and the new one.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b> : pointer to a <i>CRC_HandleTypeDef</i> structure that contains the configuration information for CRC</li> <li>• <b>pBuffer</b> : pointer to the buffer containing the data to be computed</li> <li>• <b>BufferLength</b> : length of the buffer to be computed</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>32-bit CRC</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 8.2.6.2 HAL\_CRC\_Calculate

Function Name	<b>uint32_t HAL_CRC_Calculate ( <i>CRC_HandleTypeDef</i> * hcrc,                                   uint32_t pBuffer, uint32_t BufferLength)</b>
---------------	---

Function Description	Computes the 32-bit CRC of 32-bit data buffer independently of the previous CRC value.
Parameters	<ul style="list-style-type: none"><li><b>hcrc</b> : pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC</li><li><b>pBuffer</b> : Pointer to the buffer containing the data to be computed</li><li><b>BufferLength</b> : Length of the buffer to be computed</li></ul>
Return values	<ul style="list-style-type: none"><li><b>32-bit CRC</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 8.2.7 Peripheral State functions

### 8.2.7.1 HAL\_CRC\_GetState

Function Name	<b>HAL_CRC_StateTypeDef HAL_CRC_GetState (</b> <b>CRC_HandleTypeDef * hcrc)</b>
Function Description	Returns the CRC state.
Parameters	<ul style="list-style-type: none"><li><b>hcrc</b> : pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC</li></ul>
Return values	<ul style="list-style-type: none"><li><b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 8.3 CRC Firmware driver defines

### 8.3.1 CRC

CRC

## 9 HAL CRYP Generic Driver

### 9.1 CRYP Firmware driver registers structures

#### 9.1.1 CRYP\_HandleTypeDefDef

*CRYP\_HandleTypeDefDef* is defined in the `stm32f4xx_hal_cryp.h`

##### Data Fields

- *CRYP\_InitTypeDef Init*
- *uint8\_t \* pCrypInBuffPtr*
- *uint8\_t \* pCrypOutBuffPtr*
- *\_IO uint16\_t CrypInCount*
- *\_IO uint16\_t CrypOutCount*
- *HAL\_StatusTypeDef Status*
- *HAL\_PhaseTypeDef Phase*
- *DMA\_HandleTypeDef \* hdmain*
- *DMA\_HandleTypeDef \* hdmaout*
- *HAL\_LockTypeDef Lock*
- *\_IO HAL\_CRYP\_STATETypeDef State*

##### Field Documentation

- *CRYP\_InitTypeDef CRYP\_HandleTypeDefDef::Init*
  - CRYP required parameters
- *uint8\_t\* CRYP\_HandleTypeDefDef::pCrypInBuffPtr*
  - Pointer to CRYP processing (encryption, decryption,...) buffer
- *uint8\_t\* CRYP\_HandleTypeDefDef::pCrypOutBuffPtr*
  - Pointer to CRYP processing (encryption, decryption,...) buffer
- *\_IO uint16\_t CRYP\_HandleTypeDefDef::CrypInCount*
  - Counter of inputed data
- *\_IO uint16\_t CRYP\_HandleTypeDefDef::CrypOutCount*
  - Counter of outputed data
- *HAL\_StatusTypeDef CRYP\_HandleTypeDefDef::Status*
  - CRYP peripheral status
- *HAL\_PhaseTypeDef CRYP\_HandleTypeDefDef::Phase*
  - CRYP peripheral phase
- *DMA\_HandleTypeDef\* CRYP\_HandleTypeDefDef::hdmain*
  - CRYP In DMA handle parameters
- *DMA\_HandleTypeDef\* CRYP\_HandleTypeDefDef::hdmaout*
  - CRYP Out DMA handle parameters
- *HAL\_LockTypeDef CRYP\_HandleTypeDefDef::Lock*
  - CRYP locking object
- *\_IO HAL\_CRYP\_STATETypeDef CRYP\_HandleTypeDefDef::State*
  - CRYP peripheral state

### 9.1.2 CRYP\_InitTypeDef

*CRYP\_InitTypeDef* is defined in the `stm32f4xx_hal_cryp.h`

#### Data Fields

- `uint32_t DataType`
- `uint32_t KeySize`
- `uint8_t * pKey`
- `uint8_t * pInitVect`
- `uint8_t IVSize`
- `uint8_t TagSize`
- `uint8_t * Header`
- `uint16_t HeaderSize`
- `uint8_t * pScratch`

#### Field Documentation

- `uint32_t CRYP_InitTypeDef::DataType`
  - 32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of `CRYP_Data_Type`
- `uint32_t CRYP_InitTypeDef::KeySize`
  - Used only in AES mode only : 128, 192 or 256 bit key length. This parameter can be a value of `CRYP_Key_Size`
- `uint8_t* CRYP_InitTypeDef::pKey`
  - The key used for encryption/decryption
- `uint8_t* CRYP_InitTypeDef::pInitVect`
  - The initialization vector used also as initialization counter in CTR mode
- `uint8_t CRYP_InitTypeDef::IVSize`
  - The size of initialization vector. This parameter (called nonce size in CCM) is used only in AES-128/192/256 encryption/decryption CCM mode
- `uint8_t CRYP_InitTypeDef::TagSize`
  - The size of returned authentication TAG. This parameter is used only in AES-128/192/256 encryption/decryption CCM mode
- `uint8_t* CRYP_InitTypeDef::Header`
  - The header used in GCM and CCM modes
- `uint16_t CRYP_InitTypeDef::HeaderSize`
  - The size of header buffer in bytes
- `uint8_t* CRYP_InitTypeDef::pScratch`
  - Scratch buffer used to append the header. It's size must be equal to header size + 21 bytes. This parameter is used only in AES-128/192/256 encryption/decryption CCM mode

### 9.1.3 CRYP\_TypeDef

*CRYP\_TypeDef* is defined in the `stm32f439xx.h`

#### Data Fields

- `__IO uint32_t CR`

- `__IO uint32_t SR`
- `__IO uint32_t DR`
- `__IO uint32_t DOUT`
- `__IO uint32_t DMACR`
- `__IO uint32_t IMSCR`
- `__IO uint32_t RISR`
- `__IO uint32_t MISR`
- `__IO uint32_t K0LR`
- `__IO uint32_t K0RR`
- `__IO uint32_t K1LR`
- `__IO uint32_t K1RR`
- `__IO uint32_t K2LR`
- `__IO uint32_t K2RR`
- `__IO uint32_t K3LR`
- `__IO uint32_t K3RR`
- `__IO uint32_t IV0LR`
- `__IO uint32_t IV0RR`
- `__IO uint32_t IV1LR`
- `__IO uint32_t IV1RR`
- `__IO uint32_t CSGCMCCM0R`
- `__IO uint32_t CSGCMCCM1R`
- `__IO uint32_t CSGCMCCM2R`
- `__IO uint32_t CSGCMCCM3R`
- `__IO uint32_t CSGCMCCM4R`
- `__IO uint32_t CSGCMCCM5R`
- `__IO uint32_t CSGCMCCM6R`
- `__IO uint32_t CSGCMCCM7R`
- `__IO uint32_t CSGCM0R`
- `__IO uint32_t CSGCM1R`
- `__IO uint32_t CSGCM2R`
- `__IO uint32_t CSGCM3R`
- `__IO uint32_t CSGCM4R`
- `__IO uint32_t CSGCM5R`
- `__IO uint32_t CSGCM6R`
- `__IO uint32_t CSGCM7R`

### Field Documentation

- `__IO uint32_t CRYP_TypeDef::CR`
  - CRYP control register, Address offset: 0x00
- `__IO uint32_t CRYP_TypeDef::SR`
  - CRYP status register, Address offset: 0x04
- `__IO uint32_t CRYP_TypeDef::DR`
  - CRYP data input register, Address offset: 0x08
- `__IO uint32_t CRYP_TypeDef::DOUT`
  - CRYP data output register, Address offset: 0x0C
- `__IO uint32_t CRYP_TypeDef::DMACR`
  - CRYP DMA control register, Address offset: 0x10
- `__IO uint32_t CRYP_TypeDef::IMSCR`
  - CRYP interrupt mask set/clear register, Address offset: 0x14
- `__IO uint32_t CRYP_TypeDef::RISR`

- CRYP raw interrupt status register, Address offset: 0x18
- **`__IO uint32_t CRYP_TypeDef::MISR`**
  - CRYP masked interrupt status register, Address offset: 0x1C
- **`__IO uint32_t CRYP_TypeDef::K0LR`**
  - CRYP key left register 0, Address offset: 0x20
- **`__IO uint32_t CRYP_TypeDef::K0RR`**
  - CRYP key right register 0, Address offset: 0x24
- **`__IO uint32_t CRYP_TypeDef::K1LR`**
  - CRYP key left register 1, Address offset: 0x28
- **`__IO uint32_t CRYP_TypeDef::K1RR`**
  - CRYP key right register 1, Address offset: 0x2C
- **`__IO uint32_t CRYP_TypeDef::K2LR`**
  - CRYP key left register 2, Address offset: 0x30
- **`__IO uint32_t CRYP_TypeDef::K2RR`**
  - CRYP key right register 2, Address offset: 0x34
- **`__IO uint32_t CRYP_TypeDef::K3LR`**
  - CRYP key left register 3, Address offset: 0x38
- **`__IO uint32_t CRYP_TypeDef::K3RR`**
  - CRYP key right register 3, Address offset: 0x3C
- **`__IO uint32_t CRYP_TypeDef::IV0LR`**
  - CRYP initialization vector left-word register 0, Address offset: 0x40
- **`__IO uint32_t CRYP_TypeDef::IV0RR`**
  - CRYP initialization vector right-word register 0, Address offset: 0x44
- **`__IO uint32_t CRYP_TypeDef::IV1LR`**
  - CRYP initialization vector left-word register 1, Address offset: 0x48
- **`__IO uint32_t CRYP_TypeDef::IV1RR`**
  - CRYP initialization vector right-word register 1, Address offset: 0x4C
- **`__IO uint32_t CRYP_TypeDef::CSGCMCCM0R`**
  - CRYP GCM/GMAC or CCM/CMAC context swap register 0, Address offset: 0x50
- **`__IO uint32_t CRYP_TypeDef::CSGCMCCM1R`**
  - CRYP GCM/GMAC or CCM/CMAC context swap register 1, Address offset: 0x54
- **`__IO uint32_t CRYP_TypeDef::CSGCMCCM2R`**
  - CRYP GCM/GMAC or CCM/CMAC context swap register 2, Address offset: 0x58
- **`__IO uint32_t CRYP_TypeDef::CSGCMCCM3R`**
  - CRYP GCM/GMAC or CCM/CMAC context swap register 3, Address offset: 0x5C
- **`__IO uint32_t CRYP_TypeDef::CSGCMCCM4R`**
  - CRYP GCM/GMAC or CCM/CMAC context swap register 4, Address offset: 0x60
- **`__IO uint32_t CRYP_TypeDef::CSGCMCCM5R`**
  - CRYP GCM/GMAC or CCM/CMAC context swap register 5, Address offset: 0x64
- **`__IO uint32_t CRYP_TypeDef::CSGCMCCM6R`**
  - CRYP GCM/GMAC or CCM/CMAC context swap register 6, Address offset: 0x68
- **`__IO uint32_t CRYP_TypeDef::CSGCMCCM7R`**
  - CRYP GCM/GMAC or CCM/CMAC context swap register 7, Address offset: 0x6C
- **`__IO uint32_t CRYP_TypeDef::CSGCM0R`**
  - CRYP GCM/GMAC context swap register 0, Address offset: 0x70

- `_IO uint32_t CRYP_TypeDef::CSGCM1R`  
– CRYP GCM/GMAC context swap register 1, Address offset: 0x74
- `_IO uint32_t CRYP_TypeDef::CSGCM2R`  
– CRYP GCM/GMAC context swap register 2, Address offset: 0x78
- `_IO uint32_t CRYP_TypeDef::CSGCM3R`  
– CRYP GCM/GMAC context swap register 3, Address offset: 0x7C
- `_IO uint32_t CRYP_TypeDef::CSGCM4R`  
– CRYP GCM/GMAC context swap register 4, Address offset: 0x80
- `_IO uint32_t CRYP_TypeDef::CSGCM5R`  
– CRYP GCM/GMAC context swap register 5, Address offset: 0x84
- `_IO uint32_t CRYP_TypeDef::CSGCM6R`  
– CRYP GCM/GMAC context swap register 6, Address offset: 0x88
- `_IO uint32_t CRYP_TypeDef::CSGCM7R`  
– CRYP GCM/GMAC context swap register 7, Address offset: 0x8C

## 9.2 CRYP Firmware driver API description

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

### 9.2.1 How to use this driver

The CRYP HAL driver can be used as follows:

1. Initialize the CRYP low level resources by implementing the `HAL_CRYP_MspInit()`:
  - a. Enable the CRYP interface clock using `__CRYP_CLK_ENABLE()`
  - b. In case of using interrupts (e.g. `HAL_CRYP_AESECB_Encrypt_IT()`)
    - Configure the CRYP interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the CRYP IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In CRYP IRQ handler, call `HAL_CRYP_IRQHandler()`
  - c. In case of using DMA to control data transfer (e.g. `HAL_CRYP_AESECB_Encrypt_DMA()`)
    - Enable the DMAx interface clock using `__DMAx_CLK_ENABLE()`
    - Configure and enable two DMA streams one for managing data transfer from memory to peripheral (input stream) and another stream for managing data transfer from peripheral to memory (output stream)
    - Associate the initialized DMA handle to the CRYP DMA handle using `__HAL_LINKDMA()`
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream `HAL_NVIC_SetPriority()` and `HAL_NVIC_EnableIRQ()`
2. Initialize the CRYP HAL using `HAL_CRYP_Init()`. This function configures mainly:
  - a. The data type: 1-bit, 8-bit, 16-bit and 32-bit
  - b. The key size: 128, 192 and 256. This parameter is relevant only for AES
  - c. The encryption/decryption key. Its size depends on the algorithm used for encryption/decryption
  - d. The initialization vector (counter). It is not used ECB mode.
3. Three processing (encryption/decryption) functions are available:
  - a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished, e.g. `HAL_CRYP_AESCBC_Encrypt()`

- b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL\_CRYP\_AESCBC\_Encrypt\_IT()
  - c. DMA mode: encryption and decryption APIs are not blocking functions i.e. the data transfer is ensured by DMA, e.g. HAL\_CRYP\_AESCBC\_Encrypt\_DMA()
4. When the processing function is called at first time after HAL\_CRYP\_Init() the CRYP peripheral is initialized and processes the buffer in input. At second call, the processing function performs an append of the already processed buffer. When a new data block is to be processed, call HAL\_CRYP\_Init() then the processing function.
  5. Call HAL\_CRYP\_DeInit() to deinitialize the CRYP peripheral.

## 9.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRYP according to the specified parameters in the CRYP\_InitTypeDef and creates the associated handle
- Deinitialize the CRYP peripheral
- Initialize the CRYP MSP
- Deinitialize CRYP MSP
- [\*\*HAL\\_CRYP\\_Init\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_MspDeInit\(\)\*\*](#)

## 9.2.3 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES-128/192/256 using chaining modes
- Decrypt ciphertext using AES-128/192/256 using chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode
- [\*\*HAL\\_CRYP\\_AESECB\\_Encrypt\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCBC\\_Encrypt\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCTR\\_Encrypt\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESECB\\_Decrypt\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCBC\\_Decrypt\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCTR\\_Decrypt\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESECB\\_Encrypt\\_IT\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCBC\\_Encrypt\\_IT\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCTR\\_Encrypt\\_IT\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESECB\\_Decrypt\\_IT\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCBC\\_Decrypt\\_IT\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCTR\\_Decrypt\\_IT\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESECB\\_Encrypt\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCBC\\_Encrypt\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCTR\\_Encrypt\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESECB\\_Decrypt\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_CRYP\\_AESCBC\\_Decrypt\\_DMA\(\)\*\*](#)

- [\*HAL\\_CRYPT\\_AESCTR\\_Decrypt\\_DMA\(\)\*](#)

## 9.2.4 DES processing functions

This section provides functions allowing to:

- Encrypt plaintext using DES using ECB or CBC chaining modes
- Decrypt ciphertext using ECB or CBC chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode
- [\*HAL\\_CRYPT\\_DESECB\\_encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Decrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Decrypt\\_DMA\(\)\*](#)

## 9.2.5 TDES processing functions

This section provides functions allowing to:

- Encrypt plaintext using TDES based on ECB or CBC chaining modes
- Decrypt ciphertext using TDES based on ECB or CBC chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode
- [\*HAL\\_CRYPT\\_TDESECB\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Decrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Decrypt\\_DMA\(\)\*](#)

## 9.2.6 DMA callback functions

This section provides DMA callback functions:

- DMA Input data transfer complete
- DMA Output data transfer complete
- DMA error
- [\*HAL\\_CRYP\\_InCpltCallback\(\)\*](#)
- [\*HAL\\_CRYP\\_OutCpltCallback\(\)\*](#)
- [\*HAL\\_CRYP\\_ErrorCallback\(\)\*](#)

## 9.2.7 CRYP IRQ handler management

This section provides CRYP IRQ handler function.

- [\*HAL\\_CRYP\\_IRQHandler\(\)\*](#)

## 9.2.8 Peripheral State functions

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

- [\*HAL\\_CRYP\\_GetState\(\)\*](#)

## 9.2.9 Initialization and de-initialization functions

### 9.2.9.1 HAL\_CRYP\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_Init ( <i>CRYP_HandleTypeDef</i> * <i>hcryp</i>)</b>
Function Description	Initializes the CRYP according to the specified parameters in the <i>CRYP_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a <i>CRYP_HandleTypeDef</i> structure that contains the configuration information for CRYP module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.9.2 HAL\_CRYP\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DeInit ( <i>CRYP_HandleTypeDef</i> * <i>hcryp</i>)</b>
Function Description	Deinitializes the CRYP peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a <i>CRYP_HandleTypeDef</i> structure that contains the configuration information for CRYP module</li> </ul>

---

Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.9.3 HAL\_CRYP\_MspInit

Function Name	<b>void HAL_CRYP_MspInit ( <i>CRYP_HandleTypeDef</i> * hcryp)</b>
Function Description	Initializes the CRYP MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.9.4 HAL\_CRYP\_MspDeInit

Function Name	<b>void HAL_CRYP_MspDeInit ( <i>CRYP_HandleTypeDef</i> * hcryp)</b>
Function Description	Deinitializes CRYP MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 9.2.10 AES processing functions

### 9.2.10.1 HAL\_CRYP\_AESECB\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt ( <i>CRYP_HandleTypeDef</i> * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</b>
---------------	---

Function Description	Initializes the CRYP peripheral in AES ECB encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li><b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 9.2.10.2 HAL\_CRYP\_AESCBC\_Encrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</code> <code>Size, uint8_t * pCypherData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li><b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 9.2.10.3 HAL\_CRYP\_AESCTR\_Encrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</code> <code>Size, uint8_t * pCypherData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> </ul>

---

	<ul style="list-style-type: none"> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li><b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 9.2.10.4 HAL\_CRYP\_AESECB\_Decrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 9.2.10.5 HAL\_CRYP\_AESCBC\_Decrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Timeout</b> : Specify Timeout value</li> </ul>

---

Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.10.6 HAL\_CRYP\_AESCTR\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</b> <b>uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</b>
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.10.7 HAL\_CRYP\_AESECB\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.10.8 HAL\_CRYP\_AESCBC\_Encrypt\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_IT ( CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.10.9 HAL\_CRYP\_AESCTR\_Encrypt\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_IT ( CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.10.10 HAL\_CRYP\_AESECB\_Decrypt\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_IT (     <b>CRYP_HandleTypeDef</b> * hcryp, uint8_t * pCypherData,     uint16_t Size, uint8_t * pPlainData)</code>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.10.11 HAL\_CRYP\_AESCBC\_Decrypt\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_IT (     <b>CRYP_HandleTypeDef</b> * hcryp, uint8_t * pCypherData,     uint16_t Size, uint8_t * pPlainData)</code>
Function Description	Initializes the CRYP peripheral in AES CBC decryption mode using IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.10.12 HAL\_CRYP\_AESCTR\_Decrypt\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_IT (     <b>CRYP_HandleTypeDef</b> * hcryp, uint8_t * pCypherData,</code>
---------------	---

	<b>uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.10.13 HAL\_CRYP\_AESECB\_Encrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_DMA (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in AES ECB encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.10.14 HAL\_CRYP\_AESCBC\_Encrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_DMA (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that</li> </ul>

	contains the configuration information for CRYP module
• <b>pPlainData</b> : Pointer to the plaintext buffer	
• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.	
• <b>pCypherData</b> : Pointer to the ciphertext buffer	
Return values	• <b>HAL status</b>
Notes	• None.

### 9.2.10.15 HAL\_CRYP\_AESCTR\_Encrypt\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_DMA (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</code> <code>Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16.</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 9.2.10.16 HAL\_CRYP\_AESECB\_Decrypt\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_DMA (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</code> <code>uint16_t Size, uint8_t * pPlainData)</code>
Function Description	Initializes the CRYP peripheral in AES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> </ul>

---

Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.10.17 HAL\_CRYP\_AESCBC\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_DMA (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</b> <b>uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> </ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.10.18 HAL\_CRYP\_AESCTR\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_DMA (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</b> <b>uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in AES CTR decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 16 bytes</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> </ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 9.2.11 DES processing functions

### 9.2.11.1 HAL\_CRYP\_DESECB\_Encrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</code> <code>Size, uint8_t * pCypherData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in DES ECB encryption mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.11.2 HAL\_CRYP\_DESECB\_Decrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</code> <code>Size, uint8_t * pCypherData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.11.3 HAL\_CRYP\_DESCBC\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData, uint32_t Timeout)</b>
Function Description	Initializes the CRYP peripheral in DES CBC encryption mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 9.2.11.4 HAL\_CRYP\_DESCBC\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData, uint32_t Timeout)</b>
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 9.2.11.5 HAL\_CRYP\_DESECB\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in DES ECB encryption mode

---

	using IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.11.6 HAL\_CRYP\_DESCBC\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in DES CBC encryption mode using interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.11.7 HAL\_CRYP\_DESECB\_Decrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</b> <b>uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode using IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> </ul>

- **pCypherData** : Pointer to the ciphertext buffer
  - **HAL status**
  - None.
- Return values
- Notes

### 9.2.11.8 HAL\_CRYP\_DESCBC\_Decrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</b> <b>uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode using interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 9.2.11.9 HAL\_CRYP\_DESECB\_Encrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt_DMA (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in DES ECB encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 9.2.11.10 HAL\_CRYP\_DESCBC\_Encrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt_DMA (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in DES CBC encryption mode using DMA.
Parameters	<ul style="list-style-type: none"><li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li><li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li><li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li><li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li></ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 9.2.11.11 HAL\_CRYP\_DESECB\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt_DMA (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"><li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li><li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li><li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li><li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li></ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 9.2.11.12 HAL\_CRYP\_DESCBC\_Decrypt\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</code>
Function Description	Initializes the CRYP peripheral in DES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 9.2.12 TDES processing functions

### 9.2.12.1 HAL\_CRYP\_TDESECB\_Encrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in TDES ECB encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li><b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 9.2.12.2 HAL\_CRYP\_TDESECB\_Decrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</code>
---------------	---

	<b><code>uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</code></b>
Function Description	Initializes the CRYP peripheral in TDES ECB decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.12.3 HAL\_CRYP\_TDESCBC\_Encrypt

Function Name	<b><code>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Encrypt (</code></b>
	<b><code>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</code></b>
	<b><code>Size, uint8_t * pCypherData, uint32_t Timeout)</code></b>
Function Description	Initializes the CRYP peripheral in TDES CBC encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.12.4 HAL\_CRYP\_TDESCBC\_Decrypt

Function Name	<b><code>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt (</code></b>
	<b><code>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</code></b>
	<b><code>uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</code></b>
Function Description	Initializes the CRYP peripheral in TDES CBC decryption mode then decrypted pCypherData.

Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 9.2.12.5 HAL\_CRYP\_TDESECB\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in TDES ECB encryption mode using interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 9.2.12.6 HAL\_CRYP\_TDESCBC\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Encrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</b> <b>Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYP peripheral in TDES CBC encryption mode.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li><b>pPlainData</b> : Pointer to the plaintext buffer</li> <li><b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li><b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>             |

### 9.2.12.7 HAL\_CRYP\_TDESECB\_Decrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</b> <b>uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in TDES ECB decryption mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li><li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li><li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li><li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.12.8 HAL\_CRYP\_TDESCBC\_Decrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt_IT (</b> <b>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</b> <b>uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYP peripheral in TDES CBC decryption mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li><li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li><li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li><li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 9.2.12.9 HAL\_CRYP\_TDESECB\_Encrypt\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt_DMA (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</code> <code>Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in TDES ECB encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.12.10 HAL\_CRYP\_TDESCBC\_Encrypt\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Encrypt_DMA (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t</code> <code>Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in TDES CBC encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.12.11 HAL\_CRYP\_TDESECB\_Decrypt\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt_DMA (</code> <code>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</code>
---------------	--

	<b><code>uint16_t Size, uint8_t * pPlainData)</code></b>
Function Description	Initializes the CRYP peripheral in TDES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 9.2.12.12 HAL\_CRYP\_TDESCBC\_Decrypt\_DMA

Function Name	<b><code>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt_DMA (</code></b> <b><code>CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,</code></b> <b><code>uint16_t Size, uint8_t * pPlainData)</code></b>
Function Description	Initializes the CRYP peripheral in TDES CBC decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> <li>• <b>pCypherData</b> : Pointer to the ciphertext buffer</li> <li>• <b>Size</b> : Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pPlainData</b> : Pointer to the plaintext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 9.2.13 DMA callback functions

### 9.2.13.1 HAL\_CRYP\_InCpltCallback

Function Name	<b><code>void HAL_CRYP_InCpltCallback (</code></b> <b><code>CRYP_HandleTypeDef * hcryp)</code></b>
Function Description	Input FIFO transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> </ul>

---

Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 9.2.13.2 HAL\_CRYP\_OutCpltCallback

Function Name	<b>void HAL_CRYP_OutCpltCallback ( <i>CRYP_HandleTypeDef</i> * hcryp)</b>
Function Description	Output FIFO transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a <i>CRYP_HandleTypeDef</i> structure that contains the configuration information for CRYP module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 9.2.13.3 HAL\_CRYP\_ErrorCallback

Function Name	<b>void HAL_CRYP_ErrorCallback ( <i>CRYP_HandleTypeDef</i> * hcryp)</b>
Function Description	CRYP error callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>hcryp</b> : pointer to a <i>CRYP_HandleTypeDef</i> structure that contains the configuration information for CRYP module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 9.2.14 CRYP IRQ handler management

### 9.2.14.1 HAL\_CRYP\_IRQHandler

Function Name	<b>void HAL_CRYP_IRQHandler ( <i>CRYP_HandleTypeDef</i> * hcryp)</b>
---------------	--

Function Description	This function handles CRYP interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 9.2.15 Peripheral State functions

### 9.2.15.1 HAL\_CRYP\_GetState

Function Name	<b>HAL_CRYP_STATETypeDef HAL_CRYP_GetState (</b> <b>CRYP_HandleTypeDef * hcryp)</b>
Function Description	Returns the CRYP state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b> : pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 9.3 CRYP Firmware driver defines

### 9.3.1 CRYP

CRYP

*CRYP\_AlgoModeDirection*

- #define: **CRYP\_CR\_ALGOMODE\_DIRECTION ((uint32\_t)0x0008003C)**
- #define: **CRYP\_CR\_ALGOMODE\_TDES\_ECB\_ENCRYPT ((uint32\_t)0x00000000)**
- #define: **CRYP\_CR\_ALGOMODE\_TDES\_ECB\_DECRYPT ((uint32\_t)0x00000004)**
- #define: **CRYP\_CR\_ALGOMODE\_TDES\_CBC\_ENCRYPT ((uint32\_t)0x00000008)**

- #define: ***CRYP\_CR\_ALGOMODE\_TDES\_CBC\_DECRYPT ((uint32\_t)0x0000000C)***
- #define: ***CRYP\_CR\_ALGOMODE DES ECB ENCRYPT ((uint32\_t)0x00000010)***
- #define: ***CRYP\_CR\_ALGOMODE DES ECB DECRYPT ((uint32\_t)0x00000014)***
- #define: ***CRYP\_CR\_ALGOMODE DES CBC ENCRYPT ((uint32\_t)0x00000018)***
- #define: ***CRYP\_CR\_ALGOMODE DES CBC DECRYPT ((uint32\_t)0x0000001C)***
- #define: ***CRYP\_CR\_ALGOMODE AES ECB ENCRYPT ((uint32\_t)0x00000020)***
- #define: ***CRYP\_CR\_ALGOMODE AES ECB DECRYPT ((uint32\_t)0x00000024)***
- #define: ***CRYP\_CR\_ALGOMODE AES CBC ENCRYPT ((uint32\_t)0x00000028)***
- #define: ***CRYP\_CR\_ALGOMODE AES CBC DECRYPT ((uint32\_t)0x0000002C)***
- #define: ***CRYP\_CR\_ALGOMODE AES CTR ENCRYPT ((uint32\_t)0x00000030)***
- #define: ***CRYP\_CR\_ALGOMODE AES CTR DECRYPT ((uint32\_t)0x00000034)***

### ***CRYP\_Data\_Type***

- #define: **CRYP\_DATATYPE\_32B** ((*uint32\_t*)0x00000000)
- #define: **CRYP\_DATATYPE\_16B CRYP\_CR\_DATATYPE\_0**
- #define: **CRYP\_DATATYPE\_8B CRYP\_CR\_DATATYPE\_1**
- #define: **CRYP\_DATATYPE\_1B CRYP\_CR\_DATATYPE**

#### ***CRYP\_Flags***

- #define: **CRYP\_FLAG\_BUSY** ((*uint32\_t*)0x00000010)

*The CRYP core is currently processing a block of data or a key preparation (for AES decryption).*

- #define: **CRYP\_FLAG\_IFEM** ((*uint32\_t*)0x00000001)

*Input FIFO is empty*

- #define: **CRYP\_FLAG\_IFNF** ((*uint32\_t*)0x00000002)

*Input FIFO is not Full*

- #define: **CRYP\_FLAG\_OFNE** ((*uint32\_t*)0x00000004)

*Output FIFO is not empty*

- #define: **CRYP\_FLAG\_OFFU** ((*uint32\_t*)0x00000008)

*Output FIFO is Full*

- #define: **CRYP\_FLAG\_OUTRIS** ((*uint32\_t*)0x01000002)

*Output FIFO service raw interrupt status*

- #define: **CRYP\_FLAG\_INRIS** ((*uint32\_t*)0x01000001)

*Input FIFO service raw interrupt status*

#### ***CRYP\_Interrupt***

- #define: **CRYP\_IT\_INI** ((*uint32\_t*)**CRYP\_IMSCR\_INIM**)

*Input FIFO Interrupt*

- #define: **CRYP\_IT\_OUTI** ((*uint32\_t*)**CRYP\_IMSCR\_OUTIM**)  
*Output FIFO Interrupt*

***CRYP\_Key\_Size***

- #define: **CRYP\_KEYSIZE\_128B** ((*uint32\_t*)0x00000000)
- #define: **CRYP\_KEYSIZE\_192B** **CRYP\_CR\_KEYSIZE\_0**
- #define: **CRYP\_KEYSIZE\_256B** **CRYP\_CR\_KEYSIZE\_1**

## 10 HAL DAC Generic Driver

### 10.1 DAC Firmware driver registers structures

#### 10.1.1 DAC\_HandleTypeDef

*DAC\_HandleTypeDef* is defined in the `stm32f4xx_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

#### 10.1.2 DAC\_ChannelConfTypeDef

*DAC\_ChannelConfTypeDef* is defined in the `stm32f4xx_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 [\*DAC\\_trigger\\_selection\*](#)

- ***uint32\_t DAC\_ChannelConfTypeDef::DAC\_OutputBuffer***
  - Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of **DAC\_output\_bufifer**

### 10.1.3 DAC\_TypeDef

**DAC\_TypeDef** is defined in the stm32f439xx.h

#### Data Fields

- ***\_IO uint32\_t CR***
- ***\_IO uint32\_t SWTRIGR***
- ***\_IO uint32\_t DHR12R1***
- ***\_IO uint32\_t DHR12L1***
- ***\_IO uint32\_t DHR8R1***
- ***\_IO uint32\_t DHR12R2***
- ***\_IO uint32\_t DHR12L2***
- ***\_IO uint32\_t DHR8R2***
- ***\_IO uint32\_t DHR12RD***
- ***\_IO uint32\_t DHR12LD***
- ***\_IO uint32\_t DHR8RD***
- ***\_IO uint32\_t DOR1***
- ***\_IO uint32\_t DOR2***
- ***\_IO uint32\_t SR***

#### Field Documentation

- ***\_IO uint32\_t DAC\_TypeDef::CR***
  - DAC control register, Address offset: 0x00
- ***\_IO uint32\_t DAC\_TypeDef::SWTRIGR***
  - DAC software trigger register, Address offset: 0x04
- ***\_IO uint32\_t DAC\_TypeDef::DHR12R1***
  - DAC channel1 12-bit right-aligned data holding register, Address offset: 0x08
- ***\_IO uint32\_t DAC\_TypeDef::DHR12L1***
  - DAC channel1 12-bit left aligned data holding register, Address offset: 0x0C
- ***\_IO uint32\_t DAC\_TypeDef::DHR8R1***
  - DAC channel1 8-bit right aligned data holding register, Address offset: 0x10
- ***\_IO uint32\_t DAC\_TypeDef::DHR12R2***
  - DAC channel2 12-bit right aligned data holding register, Address offset: 0x14
- ***\_IO uint32\_t DAC\_TypeDef::DHR12L2***
  - DAC channel2 12-bit left aligned data holding register, Address offset: 0x18
- ***\_IO uint32\_t DAC\_TypeDef::DHR8R2***
  - DAC channel2 8-bit right-aligned data holding register, Address offset: 0x1C
- ***\_IO uint32\_t DAC\_TypeDef::DHR12RD***
  - Dual DAC 12-bit right-aligned data holding register, Address offset: 0x20
- ***\_IO uint32\_t DAC\_TypeDef::DHR12LD***
  - DUAL DAC 12-bit left aligned data holding register, Address offset: 0x24
- ***\_IO uint32\_t DAC\_TypeDef::DHR8RD***
  - DUAL DAC 8-bit right aligned data holding register, Address offset: 0x28
- ***\_IO uint32\_t DAC\_TypeDef::DOR1***

- DAC channel1 data output register, Address offset: 0x2C
- **`_IO uint32_t DAC_TypeDef::DOR2`**
  - DAC channel2 data output register, Address offset: 0x30
- **`_IO uint32_t DAC_TypeDef::SR`**
  - DAC status register, Address offset: 0x34

## 10.2 DAC Firmware driver API description

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

### 10.2.1 DAC Peripheral features

#### DAC Channels

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

1. DAC channel1 with DAC\_OUT1 (PA4) as output
2. DAC channel2 with DAC\_OUT2 (PA5) as output

#### DAC Triggers

Digital to Analog conversion can be non-triggered using DAC\_Trigger\_None and DAC\_OUT1/DAC\_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx\_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 can be used to generate

1. Noise wave
2. Triangle wave

## DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC\_ALIGN\_8B\_R
2. 12-bit left alignment using DAC\_ALIGN\_12B\_L
3. 12-bit right alignment using DAC\_ALIGN\_12B\_R

## DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation:  $DAC\_OUTx = VREF+ * DOR / 4095$  with DOR is the Data Output Register VEF+ is the input voltage reference (refer to the device datasheet) e.g. To set DAC\_OUT1 to 0.7V, use Assuming that VREF+ = 3.3V,  $DAC\_OUT1 = (3.3 * 868) / 4095 = 0.7V$

## DMA requests

A DMA1 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 requests are enabled using HAL\_DAC\_Start\_DMA()

DMA1 requests are mapped as following:

1. DAC channel1 : mapped on DMA1 Stream5 channel7 which must be already configured
2. DAC channel2 : mapped on DMA1 Stream6 channel7 which must be already configured For Dual mode and specific signal (Triangle and noise) generation please refer to Extension Features Driver description

## 10.2.2 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 transferred 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

### 10.2.3 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\(\)\*\*](#)

### 10.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- [\*\*HAL\\_DAC\\_Start\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_Stop\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_Start\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_Stop\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_GetValue\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_IRQHandler\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_ConvCpltCallbackCh1\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_ConvHalfCpltCallbackCh1\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_ErrorCallbackCh1\(\)\*\*](#)
- [\*\*HAL\\_DAC\\_DMAUnderrunCallbackCh1\(\)\*\*](#)

### 10.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.
- [\*HAL\\_DAC\\_ConfigChannel\(\)\*](#)
- [\*HAL\\_DAC\\_SetValue\(\)\*](#)

## 10.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.
- [\*HAL\\_DAC\\_GetState\(\)\*](#)
- [\*HAL\\_DAC\\_GetError\(\)\*](#)

## 10.2.7 Initialization and de-initialization functions

### 10.2.7.1 HAL\_DAC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Init ( <a href="#"><i>DAC_HandleTypeDef</i></a> * hdac)</b>
Function Description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.7.2 HAL\_DAC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_DAC_DeInit ( <a href="#"><i>DAC_HandleTypeDef</i></a> * hdac)</b>
Function Description	Deinitializes the DAC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.7.3 HAL\_DAC\_MspInit

Function Name	<b>void HAL_DAC_MspInit ( <i>DAC_HandleTypeDef</i> * hdac)</b>
Function Description	Initializes the DAC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 10.2.7.4 HAL\_DAC\_MspDeInit

Function Name	<b>void HAL_DAC_MspDeInit ( <i>DAC_HandleTypeDef</i> * hdac)</b>
Function Description	Deinitializes the DAC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 10.2.8 IO operation functions

### 10.2.8.1 HAL\_DAC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Start ( <i>DAC_HandleTypeDef</i> * hdac, <i>uint32_t</i> Channel)</b>
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"><li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li><li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values:</li></ul>

---

	<ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1 :</b> DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2 :</b> DAC Channel2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.8.2 HAL\_DAC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Stop ( <i>DAC_HandleTypeDef</i> * hdac, uint32_t Channel)</b>
Function Description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1 :</b> DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2 :</b> DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.8.3 HAL\_DAC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Start_DMA ( <i>DAC_HandleTypeDef</i> * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)</b>
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1 :</b> DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2 :</b> DAC Channel2 selected</li> </ul> </li> <li>• <b>pData</b> : The destination peripheral Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from memory to DAC peripheral</li> <li>• <b>Alignment</b> : Specifies the data alignment for DAC channel. This parameter can be one of the following values:</li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b>DAC_ALIGN_8B_R</b> : 8bit right data alignment selected</li> <li>- <b>DAC_ALIGN_12B_L</b> : 12bit left data alignment selected</li> <li>- <b>DAC_ALIGN_12B_R</b> : 12bit right data alignment selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 10.2.8.4 HAL\_DAC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Stop_DMA ( <i>DAC_HandleTypeDef</i> * hdac, uint32_t Channel)</b>
Function Description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1</b> : DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2</b> : DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 10.2.8.5 HAL\_DAC\_GetValue

Function Name	<b>uint32_t HAL_DAC_GetValue ( <i>DAC_HandleTypeDef</i> * hdac, uint32_t Channel)</b>
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b>DAC_CHANNEL_1</b> : DAC Channel1 selected</li> <li>- <b>DAC_CHANNEL_2</b> : DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>The selected DAC channel data output value.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.8.6 HAL\_DAC\_IRQHandler

Function Name	<b>void HAL_DAC_IRQHandler ( <i>DAC_HandleTypeDef</i> * hdac)</b>
Function Description	Handles DAC interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.8.7 HAL\_DAC\_ConvCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvCpltCallbackCh1 ( <i>DAC_HandleTypeDef</i> * hdac)</b>
Function Description	Conversion complete callback in non blocking mode for Channel1.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.8.8 HAL\_DAC\_ConvHalfCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvHalfCpltCallbackCh1 ( <i>DAC_HandleTypeDef</i> * hdac)</b>
Function Description	Conversion half DMA transfer callback in non blocking mode for Channel1.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

---

Notes	<ul style="list-style-type: none"><li>None.</li></ul>
-------	---

### 10.2.8.9 HAL\_DAC\_ErrorCallbackCh1

Function Name	<b>void HAL_DAC_ErrorCallbackCh1 ( <i>DAC_HandleTypeDef</i> * hdac)</b>
Function Description	Error DAC callback for Channel1.
Parameters	<ul style="list-style-type: none"><li><b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 10.2.8.10 HAL\_DAC\_DMADebugCallbackCh1

Function Name	<b>void HAL_DAC_DMADebugCallbackCh1 ( <i>DAC_HandleTypeDef</i> * hdac)</b>
Function Description	DMA debug DAC callback for channel1.
Parameters	<ul style="list-style-type: none"><li><b>hdac</b> : pointer to a <i>DAC_HandleTypeDef</i> structure that contains the configuration information for the specified DAC.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 10.2.9 Peripheral Control functions

### 10.2.9.1 HAL\_DAC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_DAC_ConfigChannel ( <i>DAC_HandleTypeDef</i> * hdac, <i>DAC_ChannelConfTypeDef</i> * sConfig, uint32_t Channel)</b>
---------------	--

Function Description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>sConfig</b> : DAC configuration structure.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>DAC_CHANNEL_1</b> : DAC Channel1 selected</li> <li>– <b>DAC_CHANNEL_2</b> : DAC Channel2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.9.2 HAL\_DAC\_SetValue

Function Name	<b>HAL_StatusTypeDef HAL_DAC_SetValue (</b> <b>DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)</b>
Function Description	Set the specified data holding register value for DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>DAC_CHANNEL_1</b> : DAC Channel1 selected</li> <li>– <b>DAC_CHANNEL_2</b> : DAC Channel2 selected</li> </ul> </li> <li>• <b>Alignment</b> : Specifies the data alignment. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>DAC_ALIGN_8B_R</b> : 8bit right data alignment selected</li> <li>– <b>DAC_ALIGN_12B_L</b> : 12bit left data alignment selected</li> <li>– <b>DAC_ALIGN_12B_R</b> : 12bit right data alignment selected</li> </ul> </li> <li>• <b>Data</b> : Data to be loaded in the selected data holding register.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.10 Peripheral State and Errors functions

#### 10.2.10.1 HAL\_DAC\_GetState

Function Name	<b>HAL_DAC_StateTypeDef HAL_DAC_GetState (</b> <b>DAC_HandleTypeDef * hdac)</b>
Function Description	return the DAC state
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 10.2.10.2 HAL\_DAC\_GetError

Function Name	<b>uint32_t HAL_DAC_GetError (</b> <b>DAC_HandleTypeDef * hdac)</b>
Function Description	Return the DAC error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>DAC Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 10.3 DAC Firmware driver defines

### 10.3.1 DAC

DAC

*DAC\_Channel\_selection*

- #define: **DAC\_CHANNEL\_1 ((uint32\_t)0x00000000)**
  
- #define: **DAC\_CHANNEL\_2 ((uint32\_t)0x00000010)**

*DAC\_data\_alignement*

- #define: **DAC\_ALIGN\_12B\_R ((uint32\_t)0x00000000)**

- #define: **DAC\_ALIGN\_12B\_L** ((*uint32\_t*)0x00000004)
  
- #define: **DAC\_ALIGN\_8B\_R** ((*uint32\_t*)0x00000008)

#### ***DAC\_Error\_Code***

- #define: **HAL\_DAC\_ERROR\_NONE** 0x00  
*No error*
  
- #define: **HAL\_DAC\_ERROR\_DMAUNDERUNCH1** 0x01  
*DAC channel1 DAM underrun error*
  
- #define: **HAL\_DAC\_ERROR\_DMAUNDERUNCH2** 0x02  
*DAC channel2 DAM underrun error*
  
- #define: **HAL\_DAC\_ERROR\_DMA** 0x04  
*DMA error*

#### ***DAC\_flags\_definition***

- #define: **DAC\_FLAG\_DMAUDR1** ((*uint32\_t*)DAC\_SR\_DMAUDR1)
  
- #define: **DAC\_FLAG\_DMAUDR2** ((*uint32\_t*)DAC\_SR\_DMAUDR2)

#### ***DAC\_IT\_definition***

- #define: **DAC\_IT\_DMAUDR1** ((*uint32\_t*)DAC\_SR\_DMAUDR1)
  
- #define: **DAC\_IT\_DMAUDR2** ((*uint32\_t*)DAC\_SR\_DMAUDR2)

#### ***DAC\_output\_buffer***

- #define: **DAC\_OUTPUTBUFFER\_ENABLE** ((*uint32\_t*)0x00000000)

- #define: **DAC\_OUTPUTBUFFER\_DISABLE** ((uint32\_t)DAC\_CR\_BOFF1)

#### **DAC\_trigger\_selection**

- #define: **DAC\_TRIGGER\_NONE** ((uint32\_t)0x00000000)

*Conversion is automatic once the DAC1\_DHRxxxx register has been loaded, and not by external trigger*

- #define: **DAC\_TRIGGER\_T2\_TRGO** ((uint32\_t)(DAC\_CR\_TSEL1\_2 | DAC\_CR\_TEN1))

*TIM2 TRGO selected as external conversion trigger for DAC channel*

- #define: **DAC\_TRIGGER\_T4\_TRGO** ((uint32\_t)(DAC\_CR\_TSEL1\_2 | DAC\_CR\_TSEL1\_0 | DAC\_CR\_TEN1))

*TIM4 TRGO selected as external conversion trigger for DAC channel*

- #define: **DAC\_TRIGGER\_T5\_TRGO** ((uint32\_t)(DAC\_CR\_TSEL1\_1 | DAC\_CR\_TSEL1\_0 | DAC\_CR\_TEN1))

*TIM5 TRGO selected as external conversion trigger for DAC channel*

- #define: **DAC\_TRIGGER\_T6\_TRGO** ((uint32\_t)DAC\_CR\_TEN1)

*TIM6 TRGO selected as external conversion trigger for DAC channel*

- #define: **DAC\_TRIGGER\_T7\_TRGO** ((uint32\_t)(DAC\_CR\_TSEL1\_1 | DAC\_CR\_TEN1))

*TIM7 TRGO selected as external conversion trigger for DAC channel*

- #define: **DAC\_TRIGGER\_T8\_TRGO** ((uint32\_t)(DAC\_CR\_TSEL1\_0 | DAC\_CR\_TEN1))

*TIM8 TRGO selected as external conversion trigger for DAC channel*

- #define: **DAC\_TRIGGER\_EXT\_IT9** ((uint32\_t)(DAC\_CR\_TSEL1\_2 | DAC\_CR\_TSEL1\_1 | DAC\_CR\_TEN1))

*EXTI Line9 event selected as external conversion trigger for DAC channel*

- #define: **DAC\_TRIGGER\_SOFTWARE** ((uint32\_t)(DAC\_CR\_TSEL1 | DAC\_CR\_TEN1))

*Conversion started by software trigger for DAC channel*



## 11 HAL DAC Extension Driver

### 11.1 DACEx Firmware driver API description

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

#### 11.1.1 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.

#### 11.1.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.
- [\*\*HAL\\_DACEx\\_DualGetValue\(\)\*\*](#)
- [\*\*HAL\\_DACEx\\_TriangleWaveGenerate\(\)\*\*](#)
- [\*\*HAL\\_DACEx\\_NoiseWaveGenerate\(\)\*\*](#)
- [\*\*HAL\\_DACEx\\_DualSetValue\(\)\*\*](#)

#### 11.1.3 Extended features functions

##### 11.1.3.1 HAL\_DACEx\_DualGetValue

Function Name	<code>uint32_t HAL_DACEx_DualGetValue ( <a href="#">DAC_HandleTypeDef</a> * hdac)</code>
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"><li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>The selected DAC channel data output value.</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 11.1.3.2 HAL\_DACEx\_TriangleWaveGenerate

Function Name	<code>HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (</code> <code>DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t</code> <code>Amplitude)</code>
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2</li> <li>• <b>Amplitude</b> : Select max triangle amplitude. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>DAC_TRIANGLEAMPLITUDE_1</b> : Select max triangle amplitude of 1</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_3</b> : Select max triangle amplitude of 3</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_7</b> : Select max triangle amplitude of 7</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_15</b> : Select max triangle amplitude of 15</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_31</b> : Select max triangle amplitude of 31</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_63</b> : Select max triangle amplitude of 63</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_127</b> : Select max triangle amplitude of 127</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_255</b> : Select max triangle amplitude of 255</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_511</b> : Select max triangle amplitude of 511</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_1023</b> : Select max triangle amplitude of 1023</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_2047</b> : Select max triangle amplitude of 2047</li> <li>– <b>DAC_TRIANGLEAMPLITUDE_4095</b> : Select max triangle amplitude of 4095</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 11.1.3.3 HAL\_DACEx\_NoiseWaveGenerate

Function Name	<code>HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (</code> <code>DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t</code> <code>Amplitude)</code>
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Channel</b> : The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2</li> <li>• <b>Amplitude</b> : Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>DAC_LFSRUNMASK_BIT0</code> : Unmask DAC channel LFSR bit0 for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS1_0</code> : Unmask DAC channel LFSR bit[1:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS2_0</code> : Unmask DAC channel LFSR bit[2:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS3_0</code> : Unmask DAC channel LFSR bit[3:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS4_0</code> : Unmask DAC channel LFSR bit[4:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS5_0</code> : Unmask DAC channel LFSR bit[5:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS6_0</code> : Unmask DAC channel LFSR bit[6:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS7_0</code> : Unmask DAC channel LFSR bit[7:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS8_0</code> : Unmask DAC channel LFSR bit[8:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS9_0</code> : Unmask DAC channel LFSR bit[9:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS10_0</code> : Unmask DAC channel LFSR bit[10:0] for noise wave generation</li> <li>– <code>DAC_LFSRUNMASK_BITS11_0</code> : Unmask DAC channel LFSR bit[11:0] for noise wave generation</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 11.1.3.4 HAL\_DACEx\_DualSetValue

Function Name	<code>HAL_StatusTypeDef HAL_DACEx_DualSetValue (</code> <code>DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t</code>
---------------	--

<b>Data1, uint32_t Data2)</b>	
Function Description	Set the specified data holding register value for dual DAC channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac</b> : pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li>• <b>Alignment</b> : 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</li> <li>• <b>Data1</b> : Data for DAC Channel2 to be loaded in the selected data holding register.</li> <li>• <b>Data2</b> : Data for DAC Channel1 to be loaded in the selected data holding register.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In dual mode, a unique register access is required to write in both DAC channels at the same time.</li> </ul>

## 11.2 DACEx Firmware driver defines

### 11.2.1 DACEx

DACEx

**DACEx\_Lfsrunmask\_triangleamplitude**

- #define: **DAC\_LFSRUNMASK\_BIT0 ((uint32\_t)0x00000000)**

*Unmask DAC channel LFSR bit0 for noise wave generation*

- #define: **DAC\_LFSRUNMASK\_BITS1\_0 ((uint32\_t)DAC\_CR\_MAMP1\_0)**

*Unmask DAC channel LFSR bit[1:0] for noise wave generation*

- #define: **DAC\_LFSRUNMASK\_BITS2\_0 ((uint32\_t)DAC\_CR\_MAMP1\_1)**

*Unmask DAC channel LFSR bit[2:0] for noise wave generation*

- #define: **DAC\_LFSRUNMASK\_BITS3\_0 ((uint32\_t)DAC\_CR\_MAMP1\_1 | DAC\_CR\_MAMP1\_0)**

*Unmask DAC channel LFSR bit[3:0] for noise wave generation*

- #define: **DAC\_LFSRUNMASK\_BITS4\_0 ((uint32\_t)DAC\_CR\_MAMP1\_2)**

*Unmask DAC channel LFSR bit[4:0] for noise wave generation*

- #define: **DAC\_LFSRUNMASK\_BITS5\_0** ((*uint32\_t*)DAC\_CR\_MAMP1\_2 | DAC\_CR\_MAMP1\_0)  
*Unmask DAC channel LFSR bit[5:0] for noise wave generation*
  
- #define: **DAC\_LFSRUNMASK\_BITS6\_0** ((*uint32\_t*)DAC\_CR\_MAMP1\_2 | DAC\_CR\_MAMP1\_1)  
*Unmask DAC channel LFSR bit[6:0] for noise wave generation*
  
- #define: **DAC\_LFSRUNMASK\_BITS7\_0** ((*uint32\_t*)DAC\_CR\_MAMP1\_2 | DAC\_CR\_MAMP1\_1 | DAC\_CR\_MAMP1\_0)  
*Unmask DAC channel LFSR bit[7:0] for noise wave generation*
  
- #define: **DAC\_LFSRUNMASK\_BITS8\_0** ((*uint32\_t*)DAC\_CR\_MAMP1\_3)  
*Unmask DAC channel LFSR bit[8:0] for noise wave generation*
  
- #define: **DAC\_LFSRUNMASK\_BITS9\_0** ((*uint32\_t*)DAC\_CR\_MAMP1\_3 | DAC\_CR\_MAMP1\_0)  
*Unmask DAC channel LFSR bit[9:0] for noise wave generation*
  
- #define: **DAC\_LFSRUNMASK\_BITS10\_0** ((*uint32\_t*)DAC\_CR\_MAMP1\_3 | DAC\_CR\_MAMP1\_1)  
*Unmask DAC channel LFSR bit[10:0] for noise wave generation*
  
- #define: **DAC\_LFSRUNMASK\_BITS11\_0** ((*uint32\_t*)DAC\_CR\_MAMP1\_3 | DAC\_CR\_MAMP1\_1 | DAC\_CR\_MAMP1\_0)  
*Unmask DAC channel LFSR bit[11:0] for noise wave generation*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_1** ((*uint32\_t*)0x00000000)  
*Select max triangle amplitude of 1*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_3** ((*uint32\_t*)DAC\_CR\_MAMP1\_0)  
*Select max triangle amplitude of 3*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_7** ((*uint32\_t*)DAC\_CR\_MAMP1\_1)  
*Select max triangle amplitude of 7*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_15** ((*uint32\_t*)DAC\_CR\_MAMP1\_1 | DAC\_CR\_MAMP1\_0)  
*Select max triangle amplitude of 15*

- #define: **DAC\_TRIANGLEAMPLITUDE\_31** ((*uint32\_t*)DAC\_CR\_MAMP1\_2)  
*Select max triangle amplitude of 31*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_63** ((*uint32\_t*)DAC\_CR\_MAMP1\_2 | DAC\_CR\_MAMP1\_0)  
*Select max triangle amplitude of 63*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_127** ((*uint32\_t*)DAC\_CR\_MAMP1\_2 | DAC\_CR\_MAMP1\_1)  
*Select max triangle amplitude of 127*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_255** ((*uint32\_t*)DAC\_CR\_MAMP1\_2 | DAC\_CR\_MAMP1\_1 | DAC\_CR\_MAMP1\_0)  
*Select max triangle amplitude of 255*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_511** ((*uint32\_t*)DAC\_CR\_MAMP1\_3)  
*Select max triangle amplitude of 511*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_1023** ((*uint32\_t*)DAC\_CR\_MAMP1\_3 | DAC\_CR\_MAMP1\_0)  
*Select max triangle amplitude of 1023*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_2047** ((*uint32\_t*)DAC\_CR\_MAMP1\_3 | DAC\_CR\_MAMP1\_1)  
*Select max triangle amplitude of 2047*
  
- #define: **DAC\_TRIANGLEAMPLITUDE\_4095** ((*uint32\_t*)DAC\_CR\_MAMP1\_3 | DAC\_CR\_MAMP1\_1 | DAC\_CR\_MAMP1\_0)  
*Select max triangle amplitude of 4095*

#### **DACE<sub>x</sub>\_wave\_generation**

- #define: **DAC\_WAVEGENERATION\_NONE** ((*uint32\_t*)0x00000000)
  
- #define: **DAC\_WAVEGENERATION\_NOISE** ((*uint32\_t*)DAC\_CR\_WAVE1\_0)

- #define: **DAC\_WAVEGENERATION\_TRIANGLE ((uint32\_t)DAC\_CR\_WAVE1\_1)**
- #define: **DAC\_WAVE\_NOISE ((uint32\_t)DAC\_CR\_WAVE1\_0)**
- #define: **DAC\_WAVE\_TRIANGLE ((uint32\_t)DAC\_CR\_WAVE1\_1)**

## 12 HAL DMA Generic Driver

### 12.1 DMA Firmware driver registers structures

#### 12.1.1 DMA\_HandleTypeDef

*DMA\_HandleTypeDef* is defined in the `stm32f4xx_hal_dma.h`

##### Data Fields

- *DMA\_Stream\_TypeDef \* Instance*
- *DMA\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_DMA\_StateTypeDef State*
- *void \* Parent*
- *void(\* XferCpltCallback*
- *void(\* XferHalfCpltCallback*
- *void(\* XferM1CpltCallback*
- *void(\* XferErrorCallback*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- ***DMA\_Stream\_TypeDef\* DMA\_HandleTypeDef::Instance***
  - Register base address
- ***DMA\_InitTypeDef DMA\_HandleTypeDef::Init***
  - DMA communication parameters
- ***HAL\_LockTypeDef DMA\_HandleTypeDef::Lock***
  - DMA locking object
- ***\_\_IO HAL\_DMA\_StateTypeDef DMA\_HandleTypeDef::State***
  - DMA transfer state
- ***void\* DMA\_HandleTypeDef::Parent***
  - Parent object state
- ***void(\* DMA\_HandleTypeDef::XferCpltCallback)(struct \_\_DMA\_HandleTypeDef \*hdma)***
  - DMA transfer complete callback
- ***void(\* DMA\_HandleTypeDef::XferHalfCpltCallback)(struct \_\_DMA\_HandleTypeDef \*hdma)***
  - DMA Half transfer complete callback
- ***void(\* DMA\_HandleTypeDef::XferM1CpltCallback)(struct \_\_DMA\_HandleTypeDef \*hdma)***
  - DMA transfer complete Memory1 callback
- ***void(\* DMA\_HandleTypeDef::XferErrorCallback)(struct \_\_DMA\_HandleTypeDef \*hdma)***
  - DMA transfer error callback
- ***\_\_IO uint32\_t DMA\_HandleTypeDef::ErrorCode***
  - DMA Error code

## 12.1.2 DMA\_InitTypeDef

**DMA\_InitTypeDef** is defined in the `stm32f4xx_hal_dma.h`

### Data Fields

- *uint32\_t Channel*
- *uint32\_t Direction*
- *uint32\_t PeriphInc*
- *uint32\_t MemInc*
- *uint32\_t PeriphDataAlignment*
- *uint32\_t MemDataAlignment*
- *uint32\_t Mode*
- *uint32\_t Priority*
- *uint32\_t FIFOMode*
- *uint32\_t FIFOThreshold*
- *uint32\_t MemBurst*
- *uint32\_t PeriphBurst*

### Field Documentation

- ***uint32\_t DMA\_InitTypeDef::Channel***
  - Specifies the channel used for the specified stream. This parameter can be a value of [\*\*DMA\\_Channel\\_selection\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::Direction***
  - Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of [\*\*DMA\\_Data\\_transfer\\_direction\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::PeriphInc***
  - Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of [\*\*DMA\\_Peripheral\\_incremented\\_mode\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::MemInc***
  - Specifies whether the memory address register should be incremented or not. This parameter can be a value of [\*\*DMA\\_Memory\\_incremented\\_mode\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::PeriphDataAlignment***
  - Specifies the Peripheral data width. This parameter can be a value of [\*\*DMA\\_Peripheral\\_data\\_size\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::MemDataAlignment***
  - Specifies the Memory data width. This parameter can be a value of [\*\*DMA\\_Memory\\_data\\_size\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::Mode***
  - Specifies the operation mode of the DMAy Streamx. This parameter can be a value of [\*\*DMA\\_mode\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::Priority***
  - Specifies the software priority for the DMAy Streamx. This parameter can be a value of [\*\*DMA\\_Priority\\_level\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::FIFOMode***
  - Specifies if the FIFO mode or Direct mode will be used for the specified stream. This parameter can be a value of [\*\*DMA\\_FIFO\\_direct\\_mode\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::FIFOThreshold***

- Specifies the FIFO threshold level. This parameter can be a value of **DMA\_FIFO\_threshold\_level**
- **uint32\_t DMA\_InitTypeDef::MemBurst**
  - Specifies the Burst transfer configuration for the memory transfers. It specifies the amount of data to be transferred in a single non interruptable transaction.
  - This parameter can be a value of **DMA\_Memory\_burst**
- **uint32\_t DMA\_InitTypeDef::PeriphBurst**
  - Specifies the Burst transfer configuration for the peripheral transfers. It specifies the amount of data to be transferred in a single non interruptable transaction.
  - This parameter can be a value of **DMA\_Peripheral\_burst**

### 12.1.3 DMA\_Stream\_TypeDef

**DMA\_Stream\_TypeDef** is defined in the stm32f439xx.h

#### Data Fields

- **\_\_IO uint32\_t CR**
- **\_\_IO uint32\_t NDTR**
- **\_\_IO uint32\_t PAR**
- **\_\_IO uint32\_t M0AR**
- **\_\_IO uint32\_t M1AR**
- **\_\_IO uint32\_t FCR**

#### Field Documentation

- **\_\_IO uint32\_t DMA\_Stream\_TypeDef::CR**
  - DMA stream x configuration register
- **\_\_IO uint32\_t DMA\_Stream\_TypeDef::NDTR**
  - DMA stream x number of data register
- **\_\_IO uint32\_t DMA\_Stream\_TypeDef::PAR**
  - DMA stream x peripheral address register
- **\_\_IO uint32\_t DMA\_Stream\_TypeDef::M0AR**
  - DMA stream x memory 0 address register
- **\_\_IO uint32\_t DMA\_Stream\_TypeDef::M1AR**
  - DMA stream x memory 1 address register
- **\_\_IO uint32\_t DMA\_Stream\_TypeDef::FCR**
  - DMA stream x FIFO control register

### 12.1.4 DMA\_TypeDef

**DMA\_TypeDef** is defined in the stm32f439xx.h

#### Data Fields

- **\_\_IO uint32\_t LISR**
- **\_\_IO uint32\_t HISR**
- **\_\_IO uint32\_t LIFCR**
- **\_\_IO uint32\_t HIFCR**

## Field Documentation

- **`__IO uint32_t DMA_TypeDef::LISR`**
  - DMA low interrupt status register, Address offset: 0x00
- **`__IO uint32_t DMA_TypeDef::HISR`**
  - DMA high interrupt status register, Address offset: 0x04
- **`__IO uint32_t DMA_TypeDef::LIFCR`**
  - DMA low interrupt flag clear register, Address offset: 0x08
- **`__IO uint32_t DMA_TypeDef::HIFCR`**
  - DMA high interrupt flag clear register, Address offset: 0x0C

## 12.2 DMA Firmware driver API description

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

### 12.2.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Stream (except for internal SRAM/FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests .
2. For a given Stream, program the required configuration through the following parameters: Transfer Direction, Source and Destination data formats, Circular, Normal or peripheral flow control mode, Stream Priority level, Source and Destination Increment mode, FIFO mode and its Threshold (if needed), Burst mode for Source and/or Destination (if needed) using HAL\_DMA\_Init() function.

#### Polling mode IO operation

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

#### Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL\_NVIC\_SetPriority()
- Enable the DMA IRQ handler using HAL\_NVIC\_EnableIRQ()
- Use HAL\_DMA\_Start\_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL\_DMA\_IRQHandler() called under DMA\_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL\_DMA\_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of DMA handle structure).

1. Use HAL\_DMA\_GetState() function to return the DMA state and HAL\_DMA\_GetError() in case of error detection.
2. Use HAL\_DMA\_Abort() function to abort the current transfer. In Memory-to-Memory transfer mode, Circular mode is not allowed. The FIFO is used mainly to reduce bus usage and to allow data packing/unpacking: it is possible to set different Data Sizes for the Peripheral and the Memory (ie. you can set Half-Word data size for the peripheral to access its data register and set Word data size for the Memory to gain in access time. Each two half words will be packed and written in a single access to a Word in the Memory). When FIFO is disabled, it is not allowed to configure different Data Sizes for Source and Destination. In this case the Peripheral Data Size will be applied to both Source and Destination.

### DMA HAL driver macros list

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

- `_HAL_DMA_ENABLE`: Enable the specified DMA Stream.
- `_HAL_DMA_DISABLE`: Disable the specified DMA Stream.
- `_HAL_DMA_GET_FS`: Return the current DMA Stream FIFO filled level.
- `_HAL_DMA_GET_FLAG`: Get the DMA Stream pending flags.
- `_HAL_DMA_CLEAR_FLAG`: Clear the DMA Stream pending flags.
- `_HAL_DMA_ENABLE_IT`: Enable the specified DMA Stream interrupts.
- `_HAL_DMA_DISABLE_IT`: Disable the specified DMA Stream interrupts.
- `_HAL_DMA_GET_IT_SOURCE`: Check whether the specified DMA Stream interrupt has occurred or not.



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

### 12.2.2 Initialization and de-initialization functions

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

The HAL\_DMA\_Init() function follows the DMA configuration procedures as described in reference manual.

- `HAL\_DMA\_Init\(\)`
- `HAL\_DMA\_DelInit\(\)`

### 12.2.3 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\(\)\*\*\*](#)

## 12.2.4 State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code
- [\*\*\*HAL\\_DMA\\_GetState\(\)\*\*\*](#)
- [\*\*\*HAL\\_DMA\\_GetError\(\)\*\*\*](#)

## 12.2.5 Initialization and de-initialization functions

### 12.2.5.1 HAL\_DMA\_Init

Function Name	<b><i>HAL_StatusTypeDef HAL_DMA_Init ( DMA_HandleTypeDef * hdma)</i></b>
Function Description	Initializes the DMA according to the specified parameters in the DMA_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.5.2 HAL\_DMA\_DeInit

Function Name	<b><i>HAL_StatusTypeDef HAL_DMA_DeInit ( DMA_HandleTypeDef * hdma)</i></b>
Function Description	Deinitializes the DMA peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 12.2.6 I/O operation functions

### 12.2.6.1 HAL\_DMA\_Start

Function Name	<code>HAL_StatusTypeDef HAL_DMA_Start ( <i>DMA_HandleTypeDef</i> * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)</code>
Function Description	Starts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a <i>DMA_HandleTypeDef</i> structure that contains the configuration information for the specified DMA Stream.</li> <li>• <b>SrcAddress</b> : The source memory Buffer address</li> <li>• <b>DstAddress</b> : The destination memory Buffer address</li> <li>• <b>DataLength</b> : The length of data to be transferred from source to destination</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.6.2 HAL\_DMA\_Start\_IT

Function Name	<code>HAL_StatusTypeDef HAL_DMA_Start_IT ( <i>DMA_HandleTypeDef</i> * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)</code>
Function Description	Start the DMA Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a <i>DMA_HandleTypeDef</i> structure that contains the configuration information for the specified DMA Stream.</li> <li>• <b>SrcAddress</b> : The source memory Buffer address</li> <li>• <b>DstAddress</b> : The destination memory Buffer address</li> <li>• <b>DataLength</b> : The length of data to be transferred from source to destination</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.6.3 HAL\_DMA\_Abort

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Abort ( <i>DMA_HandleTypeDef</i> * hdma)</b>
Function Description	Aborts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• After disabling a DMA Stream, a check for wait until the DMA Stream is effectively disabled is added. If a Stream is disabled while a data transfer is ongoing, the current data will be transferred and the Stream will be effectively disabled only after the transfer of this single data is finished.</li> </ul>

### 12.2.6.4 HAL\_DMA\_PollForTransfer

Function Name	<b>HAL_StatusTypeDef HAL_DMA_PollForTransfer ( <i>DMA_HandleTypeDef</i> * hdma, uint32_t CompleteLevel, uint32_t Timeout)</b>
Function Description	Polling for transfer complete.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> <li>• <b>CompleteLevel</b> : Specifies the DMA level complete.</li> <li>• <b>Timeout</b> : Timeout duration.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.6.5 HAL\_DMA\_IRQHandler

Function Name	<b>void HAL_DMA_IRQHandler ( <i>DMA_HandleTypeDef</i> * hdma)</b>
Function Description	Handles DMA interrupt request.

---

Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 12.2.7 Peripheral State functions

### 12.2.7.1 HAL\_DMA\_GetState

Function Name	<b>HAL_DMA_StateTypeDef HAL_DMA_GetState (</b> <b>DMA_HandleTypeDef * hdma)</b>
Function Description	Returns the DMA state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 12.2.7.2 HAL\_DMA\_GetError

Function Name	<b>uint32_t HAL_DMA_GetError (</b> <b>DMA_HandleTypeDef * hdma)</b>
Function Description	Return the DMA error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>DMA Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 12.3 DMA Firmware driver defines

### 12.3.1 DMA

DMA

*DMA\_Channel\_selection*

- #define: **DMA\_CHANNEL\_0** ((*uint32\_t*)0x00000000)

*DMA Channel 0*

- #define: **DMA\_CHANNEL\_1** ((*uint32\_t*)0x02000000)

*DMA Channel 1*

- #define: **DMA\_CHANNEL\_2** ((*uint32\_t*)0x04000000)

*DMA Channel 2*

- #define: **DMA\_CHANNEL\_3** ((*uint32\_t*)0x06000000)

*DMA Channel 3*

- #define: **DMA\_CHANNEL\_4** ((*uint32\_t*)0x08000000)

*DMA Channel 4*

- #define: **DMA\_CHANNEL\_5** ((*uint32\_t*)0xA0000000)

*DMA Channel 5*

- #define: **DMA\_CHANNEL\_6** ((*uint32\_t*)0xC0000000)

*DMA Channel 6*

- #define: **DMA\_CHANNEL\_7** ((*uint32\_t*)0xE0000000)

*DMA Channel 7*

*DMA\_Data\_transfer\_direction*

- #define: **DMA\_PERIPH\_TO\_MEMORY** ((*uint32\_t*)0x00000000)

*Peripheral to memory direction*

- #define: **DMA\_MEMORY\_TO\_PERIPH** ((*uint32\_t*)DMA\_SxCR\_DIR\_0)

*Memory to peripheral direction*

- #define: **DMA\_MEMORY\_TO\_MEMORY** ((*uint32\_t*)DMA\_SxCR\_DIR\_1)

*Memory to memory direction*

#### **DMA\_Error\_Code**

- #define: **HAL\_DMA\_ERROR\_NONE** ((*uint32\_t*)0x00000000)  
*No error*
- #define: **HAL\_DMA\_ERROR\_TE** ((*uint32\_t*)0x00000001)  
*Transfer error*
- #define: **HAL\_DMA\_ERROR\_FE** ((*uint32\_t*)0x00000002)  
*FIFO error*
- #define: **HAL\_DMA\_ERROR\_DME** ((*uint32\_t*)0x00000004)  
*Direct Mode error*
- #define: **HAL\_DMA\_ERROR\_TIMEOUT** ((*uint32\_t*)0x00000020)  
*Timeout error*

#### **DMA\_FIFO\_direct\_mode**

- #define: **DMA\_FIFOMODE\_DISABLE** ((*uint32\_t*)0x00000000)  
*FIFO mode disable*
- #define: **DMA\_FIFOMODE\_ENABLE** ((*uint32\_t*)DMA\_SxFCR\_DMDIS)  
*FIFO mode enable*

#### **DMA\_FIFO\_threshold\_level**

- #define: **DMA\_FIFO\_THRESHOLD\_1QUARTERFULL** ((*uint32\_t*)0x00000000)  
*FIFO threshold 1 quart full configuration*
- #define: **DMA\_FIFO\_THRESHOLD\_HALFFULL** ((*uint32\_t*)DMA\_SxFCR\_FTH\_0)  
*FIFO threshold half full configuration*
- #define: **DMA\_FIFO\_THRESHOLD\_3QUARTERSFULL**  
((*uint32\_t*)DMA\_SxFCR\_FTH\_1)  
*FIFO threshold 3 quarts full configuration*
- #define: **DMA\_FIFO\_THRESHOLD\_FULL** ((*uint32\_t*)DMA\_SxFCR\_FTH)  
*FIFO threshold full configuration*

*FIFO threshold full configuration***DMA\_flag\_definitions**

- #define: **DMA\_FLAG\_FEIF0\_4** ((*uint32\_t*)0x00800001)
- #define: **DMA\_FLAG\_DMEIF0\_4** ((*uint32\_t*)0x00800004)
- #define: **DMA\_FLAG\_TEIF0\_4** ((*uint32\_t*)0x00000008)
- #define: **DMA\_FLAG\_HTIF0\_4** ((*uint32\_t*)0x00000010)
- #define: **DMA\_FLAG\_TCIF0\_4** ((*uint32\_t*)0x00000020)
- #define: **DMA\_FLAG\_FEIF1\_5** ((*uint32\_t*)0x00000040)
- #define: **DMA\_FLAG\_DMEIF1\_5** ((*uint32\_t*)0x00000100)
- #define: **DMA\_FLAG\_TEIF1\_5** ((*uint32\_t*)0x00000200)
- #define: **DMA\_FLAG\_HTIF1\_5** ((*uint32\_t*)0x00000400)
- #define: **DMA\_FLAG\_TCIF1\_5** ((*uint32\_t*)0x00000800)
- #define: **DMA\_FLAG\_FEIF2\_6** ((*uint32\_t*)0x00010000)

- #define: **DMA\_FLAG\_DMEIF2\_6** ((*uint32\_t*)0x00040000)
- #define: **DMA\_FLAG\_TEIF2\_6** ((*uint32\_t*)0x00080000)
- #define: **DMA\_FLAG\_HTIF2\_6** ((*uint32\_t*)0x00100000)
- #define: **DMA\_FLAG\_TCIF2\_6** ((*uint32\_t*)0x00200000)
- #define: **DMA\_FLAG\_FEIF3\_7** ((*uint32\_t*)0x00400000)
- #define: **DMA\_FLAG\_DMEIF3\_7** ((*uint32\_t*)0x01000000)
- #define: **DMA\_FLAG\_TEIF3\_7** ((*uint32\_t*)0x02000000)
- #define: **DMA\_FLAG\_HTIF3\_7** ((*uint32\_t*)0x04000000)
- #define: **DMA\_FLAG\_TCIF3\_7** ((*uint32\_t*)0x08000000)

#### **DMA\_Handle\_index**

- #define: **TIM\_DMA\_ID\_UPDATE** ((*uint16\_t*) 0x0)

*Index of the DMA handle used for Update DMA requests*

- #define: **TIM\_DMA\_ID\_CC1** ((*uint16\_t*) 0x1)

*Index of the DMA handle used for Capture/Compare 1 DMA requests*

- #define: **TIM\_DMA\_ID\_CC2** ((*uint16\_t*) 0x2)

*Index of the DMA handle used for Capture/Compare 2 DMA requests*

- #define: **TIM\_DMA\_ID\_CC3** ((*uint16\_t*) 0x3)

*Index of the DMA handle used for Capture/Compare 3 DMA requests*

- #define: **TIM\_DMA\_ID\_CC4** ((*uint16\_t*) 0x4)

*Index of the DMA handle used for Capture/Compare 4 DMA requests*

- #define: **TIM\_DMA\_ID\_COMMUTATION** ((*uint16\_t*) 0x5)

*Index of the DMA handle used for Commutation DMA requests*

- #define: **TIM\_DMA\_ID\_TRIGGER** ((*uint16\_t*) 0x6)

*Index of the DMA handle used for Trigger DMA requests*

#### **DMA\_interrupt\_enable\_definitions**

- #define: **DMA\_IT\_TC** ((*uint32\_t*)DMA\_SxCR\_TCIE)

- #define: **DMA\_IT\_HT** ((*uint32\_t*)DMA\_SxCR\_HTIE)

- #define: **DMA\_IT\_TE** ((*uint32\_t*)DMA\_SxCR\_TEIE)

- #define: **DMA\_IT\_DME** ((*uint32\_t*)DMA\_SxCR\_DMEIE)

- #define: **DMA\_IT\_FE** ((*uint32\_t*)0x00000080)

#### **DMA\_Memory\_burst**

- #define: **DMA\_MBURST\_SINGLE** ((*uint32\_t*)0x00000000)

- #define: **DMA\_MBURST\_INC4** ((*uint32\_t*)DMA\_SxCR\_MBURST\_0)

- #define: **DMA\_MBURST\_INC8** ((*uint32\_t*)DMA\_SxCR\_MBURST\_1)

- #define: **DMA\_MBURST\_INC16** ((*uint32\_t*)DMA\_SxCR\_MBURST)

#### **DMA\_Memory\_data\_size**

- #define: **DMA\_MDATAALIGN\_BYTE** ((*uint32\_t*)0x00000000)

*Memory data alignment: Byte*

- #define: **DMA\_MDATAALIGN\_HALFWORD** ((*uint32\_t*)DMA\_SxCR\_MSIZE\_0)

*Memory data alignment: HalfWord*

- #define: **DMA\_MDATAALIGN\_WORD** ((*uint32\_t*)DMA\_SxCR\_MSIZE\_1)

*Memory data alignment: Word*

#### **DMA\_Memory\_incremented\_mode**

- #define: **DMA\_MINC\_ENABLE** ((*uint32\_t*)DMA\_SxCR\_MINC)

*Memory increment mode enable*

- #define: **DMA\_MINC\_DISABLE** ((*uint32\_t*)0x00000000)

*Memory increment mode disable*

#### **DMA\_mode**

- #define: **DMA\_NORMAL** ((*uint32\_t*)0x00000000)

*Normal mode*

- #define: **DMA\_CIRCULAR** ((*uint32\_t*)DMA\_SxCR\_CIRC)

*Circular mode*

- #define: **DMA\_PFCTRL** ((*uint32\_t*)DMA\_SxCR\_PFCTRL)

*Peripheral flow control mode*

#### **DMA\_Peripheral\_burst**

- #define: **DMA\_PBURST\_SINGLE** ((*uint32\_t*)0x00000000)

- #define: **DMA\_PBURST\_INC4** ((*uint32\_t*)DMA\_SxCR\_PBURST\_0)

- #define: **DMA\_PBURST\_INC8** ((*uint32\_t*)DMA\_SxCR\_PBURST\_1)

- #define: **DMA\_PBURST\_INC16** ((*uint32\_t*)DMA\_SxCR\_PBURST)

#### **DMA\_Peripheral\_data\_size**

- #define: **DMA\_PDATAALIGN\_BYTE** ((*uint32\_t*)0x00000000)

*Peripheral data alignment: Byte*

- #define: **DMA\_PDATAALIGN\_HALFWORD** ((*uint32\_t*)DMA\_SxCR\_PSIZE\_0)

*Peripheral data alignment: HalfWord*

- #define: **DMA\_PDATAALIGN\_WORD** ((*uint32\_t*)DMA\_SxCR\_PSIZE\_1)

*Peripheral data alignment: Word*

#### **DMA\_Peripheral\_incremented\_mode**

- #define: **DMA\_PINC\_ENABLE** ((*uint32\_t*)DMA\_SxCR\_PINC)

*Peripheral increment mode enable*

- #define: **DMA\_PINC\_DISABLE** ((*uint32\_t*)0x00000000)

*Peripheral increment mode disable*

#### **DMA\_Priority\_level**

- #define: **DMA\_PRIORITY\_LOW** ((*uint32\_t*)0x00000000)

*Priority level: Low*

- #define: **DMA\_PRIORITY\_MEDIUM** ((*uint32\_t*)DMA\_SxCR\_PL\_0)

*Priority level: Medium*

- #define: **DMA\_PRIORITY\_HIGH** ((*uint32\_t*)DMA\_SxCR\_PL\_1)

*Priority level: High*

- #define: **DMA\_PRIORITY VERY HIGH** ((*uint32\_t*)DMA\_SxCR\_PL)  
*Priority level: Very High*

## 13 HAL DMA Extension Driver

### 13.1 DMAEx Firmware driver API description

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

#### 13.1.1 How to use this driver

The DMA Extension HAL driver can be used as follows:

1. Start a multi buffer transfer using the HAL\_DMA\_MultiBufferStart() function for polling mode or HAL\_DMA\_MultiBufferStart\_IT() for interrupt mode. In Memory-to-Memory transfer mode, Multi (Double) Buffer mode is not allowed. When Multi (Double) Buffer mode is enabled the, transfer is circular by default. In Multi (Double) buffer mode, it is possible to update the base address for the AHB memory port on the fly (DMA\_SxM0AR or DMA\_SxM1AR) when the stream is enabled.

#### 13.1.2 Extended features functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start MultiBuffer DMA transfer
- Configure the source, destination address and data length and Start MultiBuffer DMA transfer with interrupt
- Change on the fly the memory0 or memory1 address.
- [\*\*HAL\\_DMAEx\\_MultiBufferStart\(\)\*\*](#)
- [\*\*HAL\\_DMAEx\\_MultiBufferStart\\_IT\(\)\*\*](#)
- [\*\*HAL\\_DMAEx\\_ChangeMemory\(\)\*\*](#)

#### 13.1.3 Extended features functions

##### 13.1.3.1 HAL\_DMAEx\_MultiBufferStart

Function Name	<code>HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart (</code> <a href="#"><b>DMA_HandleTypeDef</b></a> <code>* hdma, uint32_t SrcAddress, uint32_t</code> <code>DstAddress, uint32_t SecondMemAddress, uint32_t</code> <code>DataLength)</code>
Function Description	Starts the multi_buffer DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> <li>• <b>SrcAddress</b> : The source memory Buffer address</li> <li>• <b>DstAddress</b> : The destination memory Buffer address</li> <li>• <b>SecondMemAddress</b> : The second memory Buffer address in case of multi buffer Transfer</li> <li>• <b>DataLength</b> : The length of data to be transferred from</li> </ul>

---

	source to destination
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.3.2 HAL\_DMAEx\_MultiBufferStart\_IT

Function Name	<b>HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart_IT (</b> <b>DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t</b> <b>DstAddress, uint32_t SecondMemAddress, uint32_t</b> <b>DataLength)</b>
Function Description	Starts the multi_buffer DMA Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> <li>• <b>SrcAddress</b> : The source memory Buffer address</li> <li>• <b>DstAddress</b> : The destination memory Buffer address</li> <li>• <b>SecondMemAddress</b> : The second memory Buffer address in case of multi buffer Transfer</li> <li>• <b>DataLength</b> : The length of data to be transferred from source to destination</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 13.1.3.3 HAL\_DMAEx\_ChangeMemory

Function Name	<b>HAL_StatusTypeDef HAL_DMAEx_ChangeMemory (</b> <b>DMA_HandleTypeDef * hdma, uint32_t Address,</b> <b>HAL_DMA_MemoryTypeDef memory)</b>
Function Description	Change the memory0 or memory1 address on the fly.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.</li> <li>• <b>Address</b> : The new address</li> <li>• <b>memory</b> : the memory to be changed, This parameter can be one of the following values: MEMORY0 / MEMORY1</li> </ul>

---

Return values

- **HAL status**

Notes

- The MEMORY0 address can be changed only when the current transfer use MEMORY1 and the MEMORY1 address can be changed only when the current transfer use MEMORY0.

## 13.2 DMAEx Firmware driver defines

### 13.2.1 DMAEx

DMAEx

## 14 HAL DMA2D Generic Driver

### 14.1 DMA2D Firmware driver registers structures

#### 14.1.1 DMA2D\_HandleTypeDef

*DMA2D\_HandleTypeDef* is defined in the `stm32f4xx_hal_dma2d.h`

##### Data Fields

- *DMA2D\_TypeDef \* Instance*
- *DMA2D\_InitTypeDef Init*
- *void(\* XferCpltCallback*
- *void(\* XferErrorCallback*
- *DMA2D\_LayerCfgTypeDef LayerCfg*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_DMA2D\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- ***DMA2D\_TypeDef\* DMA2D\_HandleTypeDef::Instance***
  - DMA2D Register base address
- ***DMA2D\_InitTypeDef DMA2D\_HandleTypeDef::Init***
  - DMA2D communication parameters
- ***void(\* DMA2D\_HandleTypeDef::XferCpltCallback)(struct \_\_DMA2D\_HandleTypeDef \*hdma2d)***
  - DMA2D transfer complete callback
- ***void(\* DMA2D\_HandleTypeDef::XferErrorCallback)(struct \_\_DMA2D\_HandleTypeDef \*hdma2d)***
  - DMA2D transfer error callback
- ***DMA2D\_LayerCfgTypeDef DMA2D\_HandleTypeDef::LayerCfg[MAX\_DMA2D\_LAYER]***
  - DMA2D Layers parameters
- ***HAL\_LockTypeDef DMA2D\_HandleTypeDef::Lock***
  - DMA2D Lock
- ***\_\_IO HAL\_DMA2D\_StateTypeDef DMA2D\_HandleTypeDef::State***
  - DMA2D transfer state
- ***\_\_IO uint32\_t DMA2D\_HandleTypeDef::ErrorCode***
  - DMA2D Error code

#### 14.1.2 DMA2D\_InitTypeDef

*DMA2D\_InitTypeDef* is defined in the `stm32f4xx_hal_dma2d.h`

##### Data Fields

- *uint32\_t Mode*

- *uint32\_t ColorMode*
- *uint32\_t OutputOffset*

#### Field Documentation

- *uint32\_t DMA2D\_InitTypeDef::Mode*
  - configures the DMA2D transfer mode. This parameter can be one value of [DMA2D\\_Mode](#)
- *uint32\_t DMA2D\_InitTypeDef::ColorMode*
  - configures the color format of the output image. This parameter can be one value of [DMA2D\\_Color\\_Mode](#)
- *uint32\_t DMA2D\_InitTypeDef::OutputOffset*
  - Specifies the Offset value. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0x3FFF.

### 14.1.3 DMA2D\_LayerCfgTypeDef

*DMA2D\_LayerCfgTypeDef* is defined in the `stm32f4xx_hal_dma2d.h`

#### Data Fields

- *uint32\_t InputOffset*
- *uint32\_t InputColorMode*
- *uint32\_t AlphaMode*
- *uint32\_t InputAlpha*

#### Field Documentation

- *uint32\_t DMA2D\_LayerCfgTypeDef::InputOffset*
  - configures the DMA2D foreground offset. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0x3FFF.
- *uint32\_t DMA2D\_LayerCfgTypeDef::InputColorMode*
  - configures the DMA2D foreground color mode . This parameter can be one value of [DMA2D\\_Input\\_Color\\_Mode](#)
- *uint32\_t DMA2D\_LayerCfgTypeDef::AlphaMode*
  - configures the DMA2D foreground alpha mode. This parameter can be one value of [DMA2D\\_ALPHA\\_MODE](#)
- *uint32\_t DMA2D\_LayerCfgTypeDef::InputAlpha*
  - Specifies the DMA2D foreground alpha value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.

### 14.1.4 DMA2D\_ColorTypeDef

*DMA2D\_ColorTypeDef* is defined in the `stm32f4xx_hal_dma2d.h`

#### Data Fields

- *uint32\_t Blue*
- *uint32\_t Green*
- *uint32\_t Red*

#### Field Documentation

- *uint32\_t DMA2D\_ColorTypeDef::Blue*
  - Configures the blue value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- *uint32\_t DMA2D\_ColorTypeDef::Green*
  - Configures the green value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- *uint32\_t DMA2D\_ColorTypeDef::Red*
  - Configures the red value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.

### 14.1.5 DMA2D\_CLUTCfgTypeDef

*DMA2D\_CLUTCfgTypeDef* is defined in the *stm32f4xx\_hal\_dma2d.h*

#### Data Fields

- *uint32\_t \* pCLUT*
- *uint32\_t CLUTColorMode*
- *uint32\_t Size*

#### Field Documentation

- *uint32\_t\* DMA2D\_CLUTCfgTypeDef::pCLUT*
  - Configures the DMA2D CLUT memory address.
- *uint32\_t DMA2D\_CLUTCfgTypeDef::CLUTColorMode*
  - configures the DMA2D CLUT color mode. This parameter can be one value of *DMA2D\_CLUT\_CM*
- *uint32\_t DMA2D\_CLUTCfgTypeDef::Size*
  - configures the DMA2D CLUT size. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.

### 14.1.6 DMA2D\_TypeDef

*DMA2D\_TypeDef* is defined in the *stm32f439xx.h*

#### Data Fields

- *\_\_IO uint32\_t CR*
- *\_\_IO uint32\_t ISR*
- *\_\_IO uint32\_t IFCR*
- *\_\_IO uint32\_t FGMAR*

- `__IO uint32_t FGOR`
- `__IO uint32_t BGMAR`
- `__IO uint32_t BGOR`
- `__IO uint32_t FGPCCR`
- `__IO uint32_t FGCOLR`
- `__IO uint32_t BGPCCR`
- `__IO uint32_t BGCOLR`
- `__IO uint32_t FGCMAR`
- `__IO uint32_t BGCMAR`
- `__IO uint32_t OPFCCR`
- `__IO uint32_t OCOLR`
- `__IO uint32_t OMAR`
- `__IO uint32_t OOR`
- `__IO uint32_t NLR`
- `__IO uint32_t LWR`
- `__IO uint32_t AMTCR`
- `uint32_t RESERVED`
- `__IO uint32_t FGCLUT`
- `__IO uint32_t BGCLUT`

### Field Documentation

- `__IO uint32_t DMA2D_TypeDef::CR`  
– DMA2D Control Register, Address offset: 0x00
- `__IO uint32_t DMA2D_TypeDef::ISR`  
– DMA2D Interrupt Status Register, Address offset: 0x04
- `__IO uint32_t DMA2D_TypeDef::IFCR`  
– DMA2D Interrupt Flag Clear Register, Address offset: 0x08
- `__IO uint32_t DMA2D_TypeDef::FGMAR`  
– DMA2D Foreground Memory Address Register, Address offset: 0x0C
- `__IO uint32_t DMA2D_TypeDef::FGOR`  
– DMA2D Foreground Offset Register, Address offset: 0x10
- `__IO uint32_t DMA2D_TypeDef::BGMAR`  
– DMA2D Background Memory Address Register, Address offset: 0x14
- `__IO uint32_t DMA2D_TypeDef::BGOR`  
– DMA2D Background Offset Register, Address offset: 0x18
- `__IO uint32_t DMA2D_TypeDef::FGPCCR`  
– DMA2D Foreground PFC Control Register, Address offset: 0x1C
- `__IO uint32_t DMA2D_TypeDef::FGCOLR`  
– DMA2D Foreground Color Register, Address offset: 0x20
- `__IO uint32_t DMA2D_TypeDef::BGPCCR`  
– DMA2D Background PFC Control Register, Address offset: 0x24
- `__IO uint32_t DMA2D_TypeDef::BGCOLR`  
– DMA2D Background Color Register, Address offset: 0x28
- `__IO uint32_t DMA2D_TypeDef::FGCMAR`  
– DMA2D Foreground CLUT Memory Address Register, Address offset: 0x2C
- `__IO uint32_t DMA2D_TypeDef::BGCMAR`  
– DMA2D Background CLUT Memory Address Register, Address offset: 0x30
- `__IO uint32_t DMA2D_TypeDef::OPFCCR`  
– DMA2D Output PFC Control Register, Address offset: 0x34
- `__IO uint32_t DMA2D_TypeDef::OCOLR`

- DMA2D Output Color Register, Address offset: 0x38
- **`_IO uint32_t DMA2D_TypeDef::OMAR`**
  - DMA2D Output Memory Address Register, Address offset: 0x3C
- **`_IO uint32_t DMA2D_TypeDef::OOR`**
  - DMA2D Output Offset Register, Address offset: 0x40
- **`_IO uint32_t DMA2D_TypeDef::NLR`**
  - DMA2D Number of Line Register, Address offset: 0x44
- **`_IO uint32_t DMA2D_TypeDef::LWR`**
  - DMA2D Line Watermark Register, Address offset: 0x48
- **`_IO uint32_t DMA2D_TypeDef::AMTCR`**
  - DMA2D AHB Master Timer Configuration Register, Address offset: 0x4C
- **`uint32_t DMA2D_TypeDef::RESERVED[236]`**
  - Reserved, 0x50-0x3FF
- **`_IO uint32_t DMA2D_TypeDef::FGCLUT[256]`**
  - DMA2D Foreground CLUT, Address offset: 400-7FF
- **`_IO uint32_t DMA2D_TypeDef::BGCLUT[256]`**
  - DMA2D Background CLUT, Address offset: 800-BFF

## 14.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

### 14.2.1 How to use this driver

1. Program the required configuration through following parameters: the Transfer Mode, the output color mode and the output offset using HAL\_DMA2D\_Init() function.
2. Program the required configuration through following parameters: the input color mode, the input color, input alpha value, alpha mode and the input offset using HAL\_DMA2D\_ConfigLayer() function for foreground or/and background layer.

#### Polling mode IO operation

- Configure the pdata, Destination and data length and Enable the transfer using HAL\_DMA2D\_Start()
- Wait for end of transfer using HAL\_DMA2D\_PollForTransfer(), at this stage user can specify the value of timeout according to his end application.

#### Interrupt mode IO operation

1. Configure the pdata, Destination and data length and Enable the transfer using HAL\_DMA2D\_Start\_IT()
2. Use HAL\_DMA2D\_IRQHandler() called under DMA2D\_IRQHandler() Interrupt subroutine
3. At the end of data transfer HAL\_DMA2D\_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of DMA2D handle structure). In Register-to-Memory transfer mode, the pdata parameter is the register color, in Memory-to-memory or memory-to-memory with pixel format conversion the pdata is the source address and

- it is the color value for the A4 or A8 mode. Configure the foreground source address, the background source address, the Destination and data length and Enable the transfer using HAL\_DMA2D\_BlendingStart() in polling mode and HAL\_DMA2D\_BlendingStart\_IT() in interrupt mode. HAL\_DMA2D\_BlendingStart() and HAL\_DMA2D\_BlendingStart\_IT() functions are used if the memory to memory with blending transfer mode is selected.
4. Optionally, configure and enable the CLUT using HAL\_DMA2D\_ConfigCLUT() HAL\_DMA2D\_EnableCLUT() functions.
  5. Optionally, configure and enable LineInterrupt using the following function: HAL\_DMA2D\_ProgramLineEvent().
  6. The transfer can be suspended, continued and aborted using the following functions: HAL\_DMA2D\_Suspend(), HAL\_DMA2D\_Resume(), HAL\_DMA2D\_Abort().
  7. To control DMA2D state you can use the following function: HAL\_DMA2D\_GetState()

### DMA2D HAL driver macros list

Below the list of most used macros in DMA2D HAL driver :

- `_HAL_DMA2D_ENABLE`: Enable the DMA2D peripheral.
- `_HAL_DMA2D_DISABLE`: Disable the DMA2D peripheral.
- `_HAL_DMA2D_GET_FLAG`: Get the DMA2D pending flags.
- `_HAL_DMA2D_CLEAR_FLAG`: Clear the DMA2D pending flags.
- `_HAL_DMA2D_ENABLE_IT`: Enable the specified DMA2D interrupts.
- `_HAL_DMA2D_DISABLE_IT`: Disable the specified DMA2D interrupts.
- `_HAL_DMA2D_GET_IT_SOURCE`: Check whether the specified DMA2D interrupt has occurred or not.



You can refer to the DMA2D HAL driver header file for more useful macros

### 14.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DMA2D
- De-initialize the DMA2D
- `HAL_DMA2D_Init()`
- `HAL_DMA2D_DelInit()`
- `HAL_DMA2D_MspInit()`
- `HAL_DMA2D_MspDelInit()`

### 14.2.3 IO operation functions

This section provides functions allowing to:

- Configure the pdata, destination address and data size and Start DMA2D transfer.
- Configure the source for foreground and background, destination address and data size and Start MultiBuffer DMA2D transfer.
- Configure the pdata, destination address and data size and Start DMA2D transfer with interrupt.
- Configure the source for foreground and background, destination address and data size and Start MultiBuffer DMA2D transfer with interrupt.

- Abort DMA2D transfer.
- Suspend DMA2D transfer.
- Continue DMA2D transfer.
- Poll for transfer complete.
- handle DMA2D interrupt request.
- [\*HAL\\_DMA2D\\_Start\(\)\*](#)
- [\*HAL\\_DMA2D\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_DMA2D\\_BlendingStart\(\)\*](#)
- [\*HAL\\_DMA2D\\_BlendingStart\\_IT\(\)\*](#)
- [\*HAL\\_DMA2D\\_Abort\(\)\*](#)
- [\*HAL\\_DMA2D\\_Suspend\(\)\*](#)
- [\*HAL\\_DMA2D\\_Resume\(\)\*](#)
- [\*HAL\\_DMA2D\\_PollForTransfer\(\)\*](#)
- [\*HAL\\_DMA2D\\_IRQHandler\(\)\*](#)

#### 14.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the DMA2D foreground or/and background parameters.
- Configure the DMA2D CLUT transfer.
- Enable DMA2D CLUT.
- Disable DMA2D CLUT.
- Configure the line watermark
- [\*HAL\\_DMA2D\\_ConfigLayer\(\)\*](#)
- [\*HAL\\_DMA2D\\_ConfigCLUT\(\)\*](#)
- [\*HAL\\_DMA2D\\_EnableCLUT\(\)\*](#)
- [\*HAL\\_DMA2D\\_DisableCLUT\(\)\*](#)
- [\*HAL\\_DMA2D\\_ProgramLineEvent\(\)\*](#)

#### 14.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to :

- Check the DMA2D state
- Get error code
- [\*HAL\\_DMA2D\\_GetState\(\)\*](#)
- [\*HAL\\_DMA2D\\_GetError\(\)\*](#)

#### 14.2.6 Initialization and Configuration functions

##### 14.2.6.1 HAL\_DMA2D\_Init

Function Name	<a href="#"><b>HAL_StatusTypeDef HAL_DMA2D_Init (</b></a> <a href="#"><b>DMA2D_HandleTypeDef * hdma2d)</b></a>
Function Description	Initializes the DMA2D according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that</li> </ul>

contains the configuration information for the DMA2D.

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>             |

#### 14.2.6.2 HAL\_DMA2D\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_DeInit (</b> <b>DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Deinitializes the DMA2D peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 14.2.6.3 HAL\_DMA2D\_MspInit

Function Name	<b>void HAL_DMA2D_MspInit (</b> <b>DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Initializes the DMA2D MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 14.2.6.4 HAL\_DMA2D\_MspDeInit

Function Name	<b>void HAL_DMA2D_MspDeInit (</b> <b>DMA2D_HandleTypeDef * hdma2d)</b>
---------------	---

**hdma2d)**

Function Description	Deinitializes the DMA2D MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**14.2.7 IO operation functions****14.2.7.1 HAL\_DMA2D\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Start (</b> <b>DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t</b> <b>DstAddress, uint32_t Width, uint32_t Heigh)</b>
Function Description	Start the DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>pdata</b> : Configure the source memory Buffer address if the memory to memory or memory to memory with pixel format conversion DMA2D mode is selected, and configure the color value if register to memory DMA2D mode is selected or the color value for the A4 or A8 mode.</li> <li>• <b>DstAddress</b> : The destination memory Buffer address.</li> <li>• <b>Width</b> : The width of data to be transferred from source to destination.</li> <li>• <b>Heigh</b> : The heigh of data to be transferred from source to destination.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**14.2.7.2 HAL\_DMA2D\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Start_IT (</b> <b>DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t</b> <b>DstAddress, uint32_t Width, uint32_t Heigh)</b>
---------------	--

Function Description	Start the DMA2D Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>pdata</b> : Configure the source memory Buffer address if the memory to memory or memory to memory with pixel format conversion DMA2D mode is selected, and configure the color value if register to memory DMA2D mode is selected or the color value for the A4 or A8 mode.</li> <li>• <b>DstAddress</b> : The destination memory Buffer address.</li> <li>• <b>Width</b> : The width of data to be transferred from source to destination.</li> <li>• <b>Heigh</b> : The heigh of data to be transferred from source to destination.</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 14.2.7.3 HAL\_DMA2D\_BlendingStart

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_BlendingStart (</b> <b>DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1,</b> <b>uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width,</b> <b>uint32_t Heigh)</b>
Function Description	Start the multi-source DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>SrcAddress1</b> : The source memory Buffer address of the foreground layer.</li> <li>• <b>SrcAddress2</b> : The source memory Buffer address of the background layer or the color value for the A4 or A8 mode.</li> <li>• <b>DstAddress</b> : The destination memory Buffer address</li> <li>• <b>Width</b> : The width of data to be transferred from source to destination.</li> <li>• <b>Heigh</b> : The heigh of data to be transferred from source to destination.</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 14.2.7.4 HAL\_DMA2D\_BlendingStart\_IT

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_BlendingStart_IT (</b> <b>DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1,</b> <b>uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width,</b> <b>uint32_t Heigh)</b>
Function Description	Start the multi-source DMA2D Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>SrcAddress1</b> : The source memory Buffer address of the foreground layer.</li> <li>• <b>SrcAddress2</b> : The source memory Buffer address of the background layer or the color value for the A4 or A8 mode.</li> <li>• <b>DstAddress</b> : The destination memory Buffer address.</li> <li>• <b>Width</b> : The width of data to be transferred from source to destination.</li> <li>• <b>Heigh</b> : The heigh of data to be transferred from source to destination.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 14.2.7.5 HAL\_DMA2D\_Abort

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Abort (</b> <b>DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Abort the DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

Notes

- None.

#### 14.2.7.6 HAL\_DMA2D\_Suspend

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Suspend (</b> <b>DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Suspend the DMA2D Transfer.

Parameters	<ul style="list-style-type: none"><li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 14.2.7.7 HAL\_DMA2D\_Resume

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Resume (</b> <b>DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Resume the DMA2D Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 14.2.7.8 HAL\_DMA2D\_PollForTransfer

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_PollForTransfer (</b> <b>DMA2D_HandleTypeDef * hdma2d, uint32_t Timeout)</b>
Function Description	Polling for transfer complete or CLUT loading.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 14.2.7.9 HAL\_DMA2D\_IRQHandler

Function Name	<b>void HAL_DMA2D_IRQHandler ( <i>DMA2D_HandleTypeDef</i> * hdma2d)</b>
Function Description	Handles DMA2D interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 14.2.8 Peripheral Control functions

### 14.2.8.1 HAL\_DMA2D\_ConfigLayer

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_ConfigLayer ( <i>DMA2D_HandleTypeDef</i> * hdma2d, uint32_t LayerIdx)</b>
Function Description	Configure the DMA2D Layer according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>LayerIdx</b> : DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 14.2.8.2 HAL\_DMA2D\_ConfigCLUT

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_ConfigCLUT ( <i>DMA2D_HandleTypeDef</i> * hdma2d, <i>DMA2D_CLUTCfgTypeDef</i> CLUTCfg, uint32_t LayerIdx)</b>
Function Description	Configure the DMA2D CLUT Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>CLUTCfg</b> : pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.</li> </ul>

- |               |   |
|---------------|---|
|               | <ul style="list-style-type: none"><li>• <b>LayerIdx</b> : DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)</li></ul> |
| Return values | <ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>   |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>   |

#### 14.2.8.3 HAL\_DMA2D\_EnableCLUT

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_EnableCLUT (</b> <b>DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)</b>
Function Description	Enable the DMA2D CLUT Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li><li>• <b>LayerIdx</b> : DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 14.2.8.4 HAL\_DMA2D\_DisableCLUT

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_DisableCLUT (</b> <b>DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)</b>
Function Description	Disable the DMA2D CLUT Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li><li>• <b>LayerIdx</b> : DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 14.2.8.5 HAL\_DMA2D\_ProgramLineEvent

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_ProgramLineEvent (</b> <b>DMA2D_HandleTypeDef * hdma2d, uint32_t Line)</b>
Function Description	Define the configuration of the line watermark .
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>Line</b> : Line Watermark configuration.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 14.2.9 Peripheral State functions

#### 14.2.9.1 HAL\_DMA2D\_GetState

Function Name	<b>HAL_DMA2D_StateTypeDef HAL_DMA2D_GetState (</b> <b>DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Return the DMA2D state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 14.2.9.2 HAL\_DMA2D\_GetError

Function Name	<b>uint32_t HAL_DMA2D_GetError (</b> <b>DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Return the DMA2D error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b> : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>DMA2D Error Code</b></li> </ul>

## 14.3 DMA2D Firmware driver defines

### 14.3.1 DMA2D

DMA2D

#### **DMA2D\_ALPHA\_MODE**

- #define: **DMA2D\_NO\_MODIF\_ALPHA** ((*uint32\_t*)0x00000000)

*No modification of the alpha channel value*

- #define: DMA2D **DMA2D\_REPLACE\_ALPHA** ((*uint32\_t*)0x00000001)

*Replace original alpha channel value by programmed alpha value*

- #define: **DMA2D\_COMBINE\_ALPHA** ((*uint32\_t*)0x00000002)

*Replace original alpha channel value by programmed alpha value with original alpha channel value*

#### **DMA2D\_CLUT\_CM**

- #define: **DMA2D\_CCM\_ARGB8888** ((*uint32\_t*)0x00000000)

*ARGB8888 DMA2D C-LUT color mode*

- #define: **DMA2D\_CCM\_RGB888** ((*uint32\_t*)0x00000001)

*RGB888 DMA2D C-LUT color mode*

#### **DMA2D\_Clut\_Size**

- #define: **DMA2D\_CLUT\_SIZE** (**DMA2D\_FGPCCR\_CS >> 8**)

*DMA2D C-LUT size*

#### **DMA2D\_Color\_Mode**

- #define: **DMA2D\_ARGB8888** ((*uint32\_t*)0x00000000)

*ARGB8888 DMA2D color mode*

- #define: **DMA2D\_RGB888** ((*uint32\_t*)0x00000001)

*RGB888 DMA2D color mode*

- #define: **DMA2D\_RGB565** ((*uint32\_t*)0x00000002)  
*RGB565 DMA2D color mode*
- #define: **DMA2D\_ARGB1555** ((*uint32\_t*)0x00000003)  
*ARGB1555 DMA2D color mode*
- #define: **DMA2D\_ARGB4444** ((*uint32\_t*)0x00000004)  
*ARGB4444 DMA2D color mode*

**DMA2D\_COLOR\_VALUE**

- #define: **COLOR\_VALUE** ((*uint32\_t*)0x000000FF)  
*color value mask*

**DMA2D\_DeadTime**

- #define: **LINE\_WATERMARK\_DMA2D\_LWR\_LW**

**DMA2D\_Error\_Code**

- #define: **HAL\_DMA2D\_ERROR\_NONE** ((*uint32\_t*)0x00000000)  
*No error*
- #define: **HAL\_DMA2D\_ERROR\_TE** ((*uint32\_t*)0x00000001)  
*Transfer error*
- #define: **HAL\_DMA2D\_ERROR\_CE** ((*uint32\_t*)0x00000002)  
*Configuration error*
- #define: **HAL\_DMA2D\_ERROR\_TIMEOUT** ((*uint32\_t*)0x00000020)  
*Timeout error*

**DMA2D\_Flag**

- #define: **DMA2D\_FLAG\_CE DMA2D\_ISR\_CEIF**  
*Configuration Error Interrupt Flag*
- #define: **DMA2D\_FLAG\_CTC DMA2D\_ISR\_CTCIF**  
*C-LUT Transfer Complete Interrupt Flag*

- #define: **DMA2D\_FLAG\_CAE** **DMA2D\_ISR\_CAEIF**  
*C-LUT Access Error Interrupt Flag*
- #define: **DMA2D\_FLAG\_TW** **DMA2D\_ISR\_TWIF**  
*Transfer Watermark Interrupt Flag*
- #define: **DMA2D\_FLAG\_TC** **DMA2D\_ISR\_TCIF**  
*Transfer Complete Interrupt Flag*
- #define: **DMA2D\_FLAG\_TE** **DMA2D\_ISR\_TEIF**  
*Transfer Error Interrupt Flag*

**DMA2D\_Input\_Color\_Mode**

- #define: **CM\_ARGB8888** ((*uint32\_t*)0x00000000)  
*ARGB8888 color mode*
- #define: **CM\_RGB888** ((*uint32\_t*)0x00000001)  
*RGB888 color mode*
- #define: **CM\_RGB565** ((*uint32\_t*)0x00000002)  
*RGB565 color mode*
- #define: **CM\_ARGB1555** ((*uint32\_t*)0x00000003)  
*ARGB1555 color mode*
- #define: **CM\_ARGB4444** ((*uint32\_t*)0x00000004)  
*ARGB4444 color mode*
- #define: **CM\_L8** ((*uint32\_t*)0x00000005)  
*L8 color mode*
- #define: **CM\_AL44** ((*uint32\_t*)0x00000006)  
*AL44 color mode*
- #define: **CM\_AL88** ((*uint32\_t*)0x00000007)

*AL88 color mode*

- #define: **CM\_L4** ((*uint32\_t*)0x00000008)  
*L4 color mode*

- #define: **CM\_A8** ((*uint32\_t*)0x00000009)  
*A8 color mode*

- #define: **CM\_A4** ((*uint32\_t*)0x0000000A)  
*A4 color mode*

**DMA2D\_Interrupts**

- #define: **DMA2D\_IT\_CE DMA2D\_CR\_CEIE**

*Configuration Error Interrupt*

- #define: **DMA2D\_IT\_CTC DMA2D\_CR\_CTCIE**

*C-LUT Transfer Complete Interrupt*

- #define: **DMA2D\_IT\_CAE DMA2D\_CR\_CAEIE**

*C-LUT Access Error Interrupt*

- #define: **DMA2D\_IT\_TW DMA2D\_CR\_TWIE**

*Transfer Watermark Interrupt*

- #define: **DMA2D\_IT\_TC DMA2D\_CR\_TCIE**

*Transfer Complete Interrupt*

- #define: **DMA2D\_IT\_TE DMA2D\_CR\_TEIE**

*Transfer Error Interrupt*

**DMA2D\_Mode**

- #define: **DMA2D\_M2M** ((*uint32\_t*)0x00000000)

*DMA2D memory to memory transfer mode*

- #define: **DMA2D\_M2M\_PFC** ((*uint32\_t*)0x00010000)

*DMA2D memory to memory with pixel format conversion transfer mode*

- #define: **DMA2D\_M2M\_BLEND ((uint32\_t)0x00020000)**  
*DMA2D memory to memory with blending transfer mode*

- #define: **DMA2D\_R2M ((uint32\_t)0x00030000)**  
*DMA2D register to memory transfer mode*

**DMA2D\_Offset**

- #define: **DMA2D\_OFFSET DMA2D\_FGOR\_LO**  
*Line Offset*

**DMA2D\_SIZE**

- #define: **DMA2D\_PIXEL (DMA2D\_NLR\_PL >> 16)**  
*DMA2D pixel per line*

- #define: **DMA2D\_LINE DMA2D\_NLR\_NL**  
*DMA2D number of line*

## 15 HAL DCMI Generic Driver

### 15.1 DCMI Firmware driver registers structures

#### 15.1.1 DCMI\_HandleTypeDef

*DCMI\_HandleTypeDef* is defined in the `stm32f4xx_hal_dcmi.h`

##### Data Fields

- *DCMI\_TypeDef \* Instance*
- *DCMI\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_DCMI\_StateTypeDef State*
- *\_\_IO uint32\_t XferCount*
- *\_\_IO uint32\_t XferSize*
- *uint32\_t XferTransferNumber*
- *uint32\_t pBuffPtr*
- *DMA\_HandleTypeDef \* DMA\_Handle*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- *DCMI\_TypeDef\* DCMI\_HandleTypeDef::Instance*
  - DCMI Register base address
- *DCMI\_InitTypeDef DCMI\_HandleTypeDef::Init*
  - DCMI parameters
- *HAL\_LockTypeDef DCMI\_HandleTypeDef::Lock*
  - DCMI locking object
- *\_\_IO HAL\_DCMI\_StateTypeDef DCMI\_HandleTypeDef::State*
  - DCMI state
- *\_\_IO uint32\_t DCMI\_HandleTypeDef::XferCount*
  - DMA transfer counter
- *\_\_IO uint32\_t DCMI\_HandleTypeDef::XferSize*
  - DMA transfer size
- *uint32\_t DCMI\_HandleTypeDef::XferTransferNumber*
  - DMA transfer number
- *uint32\_t DCMI\_HandleTypeDef::pBuffPtr*
  - Pointer to DMA output buffer
- *DMA\_HandleTypeDef\* DCMI\_HandleTypeDef::DMA\_Handle*
  - Pointer to the DMA handler
- *\_\_IO uint32\_t DCMI\_HandleTypeDef::ErrorCode*
  - DCMI Error code

#### 15.1.2 DCMI\_InitTypeDef

*DCMI\_InitTypeDef* is defined in the `stm32f4xx_hal_dcmi.h`

**Data Fields**

- *uint32\_t SynchroMode*
- *uint32\_t PCKPolarity*
- *uint32\_t VSPolarity*
- *uint32\_t HSPolarity*
- *uint32\_t CaptureRate*
- *uint32\_t ExtendedDataMode*
- *DCMI\_CodesInitTypeDef SyncroCode*
- *uint32\_t JPEGMode*

**Field Documentation**

- *uint32\_t DCMI\_InitTypeDef::SynchroMode*
  - Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of [\*DCMI\\_Synchronization\\_Mode\*](#)
- *uint32\_t DCMI\_InitTypeDef::PCKPolarity*
  - Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of [\*DCMI\\_PIXCK\\_Polarity\*](#)
- *uint32\_t DCMI\_InitTypeDef::VSPolarity*
  - Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of [\*DCMI\\_VSYNC\\_Polarity\*](#)
- *uint32\_t DCMI\_InitTypeDef::HSPolarity*
  - Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of [\*DCMI\\_HSYNC\\_Polarity\*](#)
- *uint32\_t DCMI\_InitTypeDef::CaptureRate*
  - Specifies the frequency of frame capture: All, 1/2 or 1/4. This parameter can be a value of [\*DCMI\\_Capture\\_Rate\*](#)
- *uint32\_t DCMI\_InitTypeDef::ExtendedDataMode*
  - Specifies the data width: 8-bit, 10-bit, 12-bit or 14-bit. This parameter can be a value of [\*DCMI\\_Extended\\_Data\\_Mode\*](#)
- *DCMI\_CodesInitTypeDef DCMI\_InitTypeDef::SyncroCode*
  - Specifies the code of the frame start delimiter.
- *uint32\_t DCMI\_InitTypeDef::JPEGMode*
  - Enable or Disable the JPEG mode. This parameter can be a value of [\*DCMI\\_MODE\\_JPEG\*](#)

**15.1.3 DCMI\_CodesInitTypeDef**

*DCMI\_CodesInitTypeDef* is defined in the `stm32f4xx_hal_dcmi.h`

**Data Fields**

- *uint8\_t FrameStartCode*
- *uint8\_t LineStartCode*
- *uint8\_t LineEndCode*
- *uint8\_t FrameEndCode*

---

**Field Documentation**

- ***uint8\_t DCMI\_CodesInitTypeDef::FrameStartCode***
  - Specifies the code of the frame start delimiter.
- ***uint8\_t DCMI\_CodesInitTypeDef::LineStartCode***
  - Specifies the code of the line start delimiter.
- ***uint8\_t DCMI\_CodesInitTypeDef::LineEndCode***
  - Specifies the code of the line end delimiter.
- ***uint8\_t DCMI\_CodesInitTypeDef::FrameEndCode***
  - Specifies the code of the frame end delimiter.

### 15.1.4 DCMI\_TypeDef

**DCMI\_TypeDef** is defined in the stm32f439xx.h

**Data Fields**

- ***\_IO uint32\_t CR***
- ***\_IO uint32\_t SR***
- ***\_IO uint32\_t RISR***
- ***\_IO uint32\_t IER***
- ***\_IO uint32\_t MISR***
- ***\_IO uint32\_t ICR***
- ***\_IO uint32\_t ESCR***
- ***\_IO uint32\_t ESUR***
- ***\_IO uint32\_t CWSTRTR***
- ***\_IO uint32\_t CWSIZER***
- ***\_IO uint32\_t DR***

**Field Documentation**

- ***\_IO uint32\_t DCMI\_TypeDef::CR***
  - DCMI control register 1, Address offset: 0x00
- ***\_IO uint32\_t DCMI\_TypeDef::SR***
  - DCMI status register, Address offset: 0x04
- ***\_IO uint32\_t DCMI\_TypeDef::RISR***
  - DCMI raw interrupt status register, Address offset: 0x08
- ***\_IO uint32\_t DCMI\_TypeDef::IER***
  - DCMI interrupt enable register, Address offset: 0x0C
- ***\_IO uint32\_t DCMI\_TypeDef::MISR***
  - DCMI masked interrupt status register, Address offset: 0x10
- ***\_IO uint32\_t DCMI\_TypeDef::ICR***
  - DCMI interrupt clear register, Address offset: 0x14
- ***\_IO uint32\_t DCMI\_TypeDef::ESCR***
  - DCMI embedded synchronization code register, Address offset: 0x18
- ***\_IO uint32\_t DCMI\_TypeDef::ESUR***
  - DCMI embedded synchronization unmask register, Address offset: 0x1C
- ***\_IO uint32\_t DCMI\_TypeDef::CWSTRTR***
  - DCMI crop window start, Address offset: 0x20

- `_IO uint32_t DCMI_TypeDef::CWSIZER`
  - DCMI crop window size, Address offset: 0x24
- `_IO uint32_t DCMI_TypeDef::DR`
  - DCMI data register, Address offset: 0x28

## 15.2 DCMI Firmware driver API description

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

### 15.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMI Interface. This sequence does not take into account the configuration of the camera module, which should be made before to configure and enable the DCMI to capture images.

1. Program the required configuration through following parameters: horizontal and vertical polarity, pixel clock polarity, Capture Rate, Synchronization Mode, code of the frame delimiter and data width using `HAL_DCMI_Init()` function.
2. Configure the DMA2\_Stream1 channel1 to transfer Data from DCMI DR register to the destination memory buffer.
3. Program the required configuration through following parameters: DCMI mode, destination memory Buffer address and the data length and enable capture using `HAL_DCMI_Start_DMA()` function.
4. Optionally, configure and Enable the CROP feature to select a rectangular window from the received image using `HAL_DCMI_ConfigCrop()` and `HAL_DCMI_EnableCROP()` functions
5. The capture can be stopped using `HAL_DCMI_Stop()` function.
6. To control DCMI state you can use the function `HAL_DCMI_GetState()`.

#### DCMI HAL driver macros list

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

- `_HAL_DCMI_ENABLE`: Enable the DCMI peripheral.
- `_HAL_DCMI_DISABLE`: Disable the DCMI peripheral.
- `_HAL_DCMI_GET_FLAG`: Get the DCMI pending flags.
- `_HAL_DCMI_CLEAR_FLAG`: Clear the DCMI pending flags.
- `_HAL_DCMI_ENABLE_IT`: Enable the specified DCMI interrupts.
- `_HAL_DCMI_DISABLE_IT`: Disable the specified DCMI interrupts.
- `_HAL_DCMI_GET_IT_SOURCE`: Check whether the specified DCMI interrupt has occurred or not.



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

### 15.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI
- [\*HAL\\_DCMI\\_Init\(\)\*](#)
- [\*HAL\\_DCMI\\_DelInit\(\)\*](#)
- [\*HAL\\_DCMI\\_MspInit\(\)\*](#)
- [\*HAL\\_DCMI\\_MspDelInit\(\)\*](#)

### 15.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length and Enables DCMI DMA request and enables DCMI capture
- Stop the DCMI capture.
- handle DCMI interrupt request.
- [\*HAL\\_DCMI\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_DCMI\\_Stop\(\)\*](#)
- [\*HAL\\_DCMI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_DCMI\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_LineEventCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_VsyncEventCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_FrameEventCallback\(\)\*](#)

### 15.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the CROP feature.
- Enable/Disable the CROP feature.
- [\*HAL\\_DCMI\\_ConfigCROP\(\)\*](#)
- [\*HAL\\_DCMI\\_DisableCROP\(\)\*](#)
- [\*HAL\\_DCMI\\_EnableCROP\(\)\*](#)

### 15.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DCMI state.
- Get the specific DCMI error flag.
- [\*HAL\\_DCMI\\_GetState\(\)\*](#)
- [\*HAL\\_DCMI\\_GetError\(\)\*](#)

### 15.2.6 Initialization and Configuration functions

#### 15.2.6.1 HAL\_DCMI\_Init

Function Name	<code>HAL_StatusTypeDef HAL_DCMI_Init ( <i>DCMI_HandleTypeDef</i> *hdcmi)</code>
Function Description	Initializes the DCMI according to the specified parameters in the

DCMI\_InitTypeDef and create the associated handle.

Parameters	<ul style="list-style-type: none"><li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 15.2.6.2 HAL\_DCMI\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_DelInit (</b> <b><i>DCMI_HandleTypeDef * hdcmi</i></b> )
Function Description	Deinitializes the DCMI peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"><li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 15.2.6.3 HAL\_DCMI\_MspInit

Function Name	<b>void HAL_DCMI_MspInit (</b> <b><i>DCMI_HandleTypeDef * hdcmi</i></b> )
Function Description	Initializes the DCMI MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 15.2.6.4 HAL\_DCMI\_MspDelInit

Function Name	<b>void HAL_DCMI_MspDeInit ( <i>DCMI_HandleTypeDef</i> * hdcmi)</b>
Function Description	Deinitializes the DCMI MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 15.2.7 IO operation functions

### 15.2.7.1 HAL\_DCMI\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_Start_DMA ( <i>DCMI_HandleTypeDef</i> * hdcmi, uint32_t DCMI_Mode, uint32_t pData, uint32_t Length)</b>
Function Description	Enables DCMI DMA request and enables DCMI capture.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> <li>• <b>DCMI_Mode</b> : DCMI capture mode snapshot or continuous grab.</li> <li>• <b>pData</b> : The destination memory Buffer address (LCD Frame buffer).</li> <li>• <b>Length</b> : The length of capture to be transferred.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 15.2.7.2 HAL\_DCMI\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_Stop ( <i>DCMI_HandleTypeDef</i> * hdcmi)</b>
Function Description	Disable DCMI DMA request and Disable DCMI capture.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 15.2.7.3 HAL\_DCMI\_IRQHandler

Function Name	<b>void HAL_DCMI_IRQHandler ( <i>DCMI_HandleTypeDef</i> * hdcmi)</b>
Function Description	Handles DCMI interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>(hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for the DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 15.2.7.4 HAL\_DCMI\_ErrorCallback

Function Name	<b>void HAL_DCMI_ErrorCallback ( <i>DCMI_HandleTypeDef</i> * hdcmi)</b>
Function Description	Error DCMI callback.
Parameters	<ul style="list-style-type: none"><li>• <b>(hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 15.2.7.5 HAL\_DCMI\_LineEventCallback

Function Name	<b>void HAL_DCMI_LineEventCallback ( <i>DCMI_HandleTypeDef</i> * hdcmi)</b>
Function Description	Line Event callback.
Parameters	<ul style="list-style-type: none"><li>• <b>(hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

- Notes • None.

### 15.2.7.6 HAL\_DCMI\_VsyncEventCallback

Function Name	<b>void HAL_DCMI_VsyncEventCallback ( <i>DCMI_HandleTypeDefDef * hdcmi</i>)</b>
Function Description	VSYNC Event callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDefDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 15.2.7.7 HAL\_DCMI\_FrameEventCallback

Function Name	<b>void HAL_DCMI_FrameEventCallback ( <i>DCMI_HandleTypeDefDef * hdcmi</i>)</b>
Function Description	Frame Event callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDefDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 15.2.8 Peripheral Control functions

### 15.2.8.1 HAL\_DCMI\_ConfigCROP

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_ConfigCROP ( <i>DCMI_HandleTypeDefDef * hdcmi, uint32_t X0, uint32_t Y0, uint32_t XSize, uint32_t YSize</i>)</b>
---------------	--

Function Description	Configure the DCMI CROP coordinate.
Parameters	<ul style="list-style-type: none"><li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li><li>• <b>YSize</b> : DCMI Line number</li><li>• <b>XSize</b> : DCMI Pixel per line</li><li>• <b>X0</b> : DCMI window X offset</li><li>• <b>Y0</b> : DCMI window Y offset</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 15.2.8.2 HAL\_DCMI\_DisableCROP

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_DisableCROP (</b> <i>DCMI_HandleTypeDef * hdcmi)</i>
Function Description	Disable the Crop feature.
Parameters	<ul style="list-style-type: none"><li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 15.2.8.3 HAL\_DCMI\_EnableCROP

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_EnableCROP (</b> <i>DCMI_HandleTypeDef * hdcmi)</i>
Function Description	Enable the Crop feature.
Parameters	<ul style="list-style-type: none"><li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 15.2.9 Peripheral State functions

### 15.2.9.1 HAL\_DCMI\_GetState

Function Name	<b>HAL_DCMI_StateTypeDef HAL_DCMI_GetState (</b> <b><i>DCMI_HandleTypeDef</i> * hdcmi)</b>
Function Description	Return the DCMI state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 15.2.9.2 HAL\_DCMI\_GetError

Function Name	<b>uint32_t HAL_DCMI_GetError (</b> <b><i>DCMI_HandleTypeDef</i> * hdcmi)</b>
Function Description	Return the DCMI error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b> : pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>DCMI Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 15.3 DCMI Firmware driver defines

### 15.3.1 DCMI

DCMI

***DCMI\_Capture\_Mode***

- #define: ***DCMI\_MODE\_CONTINUOUS* ((uint32\_t)0x00000000)**

*The received data are transferred continuously into the destination memory through the DMA*

- #define: ***DCMI\_MODE\_SNAPSHOT* ((uint32\_t)DCMI\_CR\_CM)**

*Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA*

#### ***DCMI\_Capture\_Rate***

- #define: ***DCMI\_CR\_ALL\_FRAME*** ((*uint32\_t*)0x00000000)

*All frames are captured*

- #define: ***DCMI\_CR\_ALTERNATE\_2\_FRAME*** ((*uint32\_t*)***DCMI\_CR\_FCRC\_0***)

*Every alternate frame captured*

- #define: ***DCMI\_CR\_ALTERNATE\_4\_FRAME*** ((*uint32\_t*)***DCMI\_CR\_FCRC\_1***)

*One frame in 4 frames captured*

#### ***DCMI\_Error\_Code***

- #define: ***HAL\_DCMI\_ERROR\_NONE*** ((*uint32\_t*)0x00000000)

*No error*

- #define: ***HAL\_DCMI\_ERROR\_OVF*** ((*uint32\_t*)0x00000001)

*Overflow error*

- #define: ***HAL\_DCMI\_ERROR\_SYNC*** ((*uint32\_t*)0x00000002)

*Synchronization error*

- #define: ***HAL\_DCMI\_ERROR\_TIMEOUT*** ((*uint32\_t*)0x00000020)

*Timeout error*

#### ***DCMI\_Extended\_Data\_Mode***

- #define: ***DCMI\_EXTEND\_DATA\_8B*** ((*uint32\_t*)0x00000000)

*Interface captures 8-bit data on every pixel clock*

- #define: ***DCMI\_EXTEND\_DATA\_10B*** ((*uint32\_t*)***DCMI\_CR\_EDM\_0***)

*Interface captures 10-bit data on every pixel clock*

- #define: ***DCMI\_EXTEND\_DATA\_12B*** ((*uint32\_t*)***DCMI\_CR\_EDM\_1***)

*Interface captures 12-bit data on every pixel clock*

- #define: **DCMI\_EXTEND\_DATA\_14B** ((*uint32\_t*)(**DCMI\_CR\_EDM\_0 | DCMI\_CR\_EDM\_1**))

*Interface captures 14-bit data on every pixel clock*

#### ***DCMI\_Flags***

- #define: **DCMI\_FLAG\_HSYNC** ((*uint32\_t*)0x2001)
- #define: **DCMI\_FLAG\_VSYNC** ((*uint32\_t*)0x2002)
- #define: **DCMI\_FLAG\_FNE** ((*uint32\_t*)0x2004)
- #define: **DCMI\_FLAG\_FRAMERI** ((*uint32\_t*)**DCMI\_RISR\_FRAME\_RIS**)
- #define: **DCMI\_FLAG\_OVFRI** ((*uint32\_t*)**DCMI\_RISR\_OVF\_RIS**)
- #define: **DCMI\_FLAG\_ERRRI** ((*uint32\_t*)**DCMI\_RISR\_ERR\_RIS**)
- #define: **DCMI\_FLAG\_VSYNCRI** ((*uint32\_t*)**DCMI\_RISR\_VSYNC\_RIS**)
- #define: **DCMI\_FLAG\_LINERI** ((*uint32\_t*)**DCMI\_RISR\_LINE\_RIS**)
- #define: **DCMI\_FLAG\_FRAMEMI** ((*uint32\_t*)0x1001)
- #define: **DCMI\_FLAG\_OVFM** ((*uint32\_t*)0x1002)
- #define: **DCMI\_FLAG\_ERRMI** ((*uint32\_t*)0x1004)

- #define: **DCMI\_FLAG\_VSYNCMI** ((*uint32\_t*)0x1008)

- #define: **DCMI\_FLAG\_LINEMI** ((*uint32\_t*)0x1010)

#### ***DCMI\_HSYNC\_Polarity***

- #define: **DCMI\_HSPOLARITY\_LOW** ((*uint32\_t*)0x00000000)

*Horizontal synchronization active Low*

- #define: **DCMI\_HSPOLARITY\_HIGH** ((*uint32\_t*)DCMI\_CR\_HSPOL)

*Horizontal synchronization active High*

#### ***DCMI\_interrupt\_sources***

- #define: **DCMI\_IT\_FRAME** ((*uint32\_t*)DCMI\_IER\_FRAME\_IE)

- #define: **DCMI\_IT\_OVF** ((*uint32\_t*)DCMI\_IER\_OVF\_IE)

- #define: **DCMI\_IT\_ERR** ((*uint32\_t*)DCMI\_IER\_ERR\_IE)

- #define: **DCMI\_IT\_VSYNC** ((*uint32\_t*)DCMI\_IER\_VSYNC\_IE)

- #define: **DCMI\_IT\_LINE** ((*uint32\_t*)DCMI\_IER\_LINE\_IE)

#### ***DCMI\_MODE\_JPEG***

- #define: **DCMI\_JPEG\_DISABLE** ((*uint32\_t*)0x00000000)

*Mode JPEG Disabled*

- #define: **DCMI\_JPEG\_ENABLE** ((*uint32\_t*)DCMI\_CR\_JPEG)

*Mode JPEG Enabled*

***DCMI\_PIXCK\_Polarity***

- #define: ***DCMI\_PCKPOLARITY\_FALLING*** ((*uint32\_t*)0x00000000)

*Pixel clock active on Falling edge*

- #define: ***DCMI\_PCKPOLARITY\_RISING*** ((*uint32\_t*)DCMI\_CR\_PCKPOL)

*Pixel clock active on Rising edge*

***DCMI\_Synchronization\_Mode***

- #define: ***DCMI\_SYNCHRO\_HARDWARE*** ((*uint32\_t*)0x00000000)

*Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals*

- #define: ***DCMI\_SYNCHRO\_EMBEDDED*** ((*uint32\_t*)DCMI\_CR\_ESS)

*Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow*

***DCMI\_VSYNC\_Polarity***

- #define: ***DCMI\_VSPOLARITY\_LOW*** ((*uint32\_t*)0x00000000)

*Vertical synchronization active Low*

- #define: ***DCMI\_VSPOLARITY\_HIGH*** ((*uint32\_t*)DCMI\_CR\_VSPOL)

*Vertical synchronization active High*

***DCMI\_Window\_Coordinate***

- #define: ***DCMI\_WINDOW\_COORDINATE*** ((*uint32\_t*)0x3FFF)

*Window coordinate*

***DCMI\_Window\_Height***

- #define: ***DCMI\_WINDOW\_HEIGHT*** ((*uint32\_t*)0x1FFF)

*Window Height*

## 16 HAL ETHERNET Generic Driver

### 16.1 ETH Firmware driver registers structures

#### 16.1.1 ETH\_HandleTypeDef

*ETH\_HandleTypeDef* is defined in the `stm32f4xx_hal_eth.h`

##### Data Fields

- *ETH\_TypeDef \* Instance*
- *ETH\_InitTypeDef Init*
- *uint32\_t LinkStatus*
- *ETH\_DMADescTypeDef \* RxDesc*
- *ETH\_DMADescTypeDef \* TxDesc*
- *ETH\_DMARxFrameInfos RxFrameInfos*
- *\_IO HAL\_ETH\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*

##### Field Documentation

- ***ETH\_TypeDef\* ETH\_HandleTypeDef::Instance***
  - Register base address
- ***ETH\_InitTypeDef ETH\_HandleTypeDef::Init***
  - Ethernet Init Configuration
- ***uint32\_t ETH\_HandleTypeDef::LinkStatus***
  - Ethernet link status
- ***ETH\_DMADescTypeDef\* ETH\_HandleTypeDef::RxDesc***
  - Rx descriptor to Get
- ***ETH\_DMADescTypeDef\* ETH\_HandleTypeDef::TxDesc***
  - Tx descriptor to Set
- ***ETH\_DMARxFrameInfos ETH\_HandleTypeDef::RxFrameInfos***
  - last Rx frame infos
- ***\_IO HAL\_ETH\_StateTypeDef ETH\_HandleTypeDef::State***
  - ETH communication state
- ***HAL\_LockTypeDef ETH\_HandleTypeDef::Lock***
  - ETH Lock

#### 16.1.2 ETH\_InitTypeDef

*ETH\_InitTypeDef* is defined in the `stm32f4xx_hal_eth.h`

##### Data Fields

- *uint32\_t AutoNegotiation*
- *uint32\_t Speed*
- *uint32\_t DuplexMode*

- *uint16\_t PhyAddress*
- *uint8\_t \* MACAddr*
- *uint32\_t RxMode*
- *uint32\_t ChecksumMode*
- *uint32\_t MedialInterface*

#### Field Documentation

- *uint32\_t ETH\_InitTypeDef::AutoNegotiation*
  - Selects or not the AutoNegotiation mode for the external PHY. The AutoNegotiation allows an automatic setting of the Speed (10/100Mbps) and the mode (half/full-duplex). This parameter can be a value of [ETH\\_AutoNegotiation](#)
- *uint32\_t ETH\_InitTypeDef::Speed*
  - Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of [ETH\\_Speed](#)
- *uint32\_t ETH\_InitTypeDef::DuplexMode*
  - Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode. This parameter can be a value of [ETH\\_Duplex\\_Mode](#)
- *uint16\_t ETH\_InitTypeDef::PhyAddress*
  - Ethernet PHY address. This parameter must be a number between Min\_Data = 0 and Max\_Data = 32
- *uint8\_t\* ETH\_InitTypeDef::MACAddr*
  - MAC Address of used Hardware: must be pointer on an array of 6 bytes
- *uint32\_t ETH\_InitTypeDef::RxMode*
  - Selects the Ethernet Rx mode: Polling mode, Interrupt mode. This parameter can be a value of [ETH\\_Rx\\_Mode](#)
- *uint32\_t ETH\_InitTypeDef::ChecksumMode*
  - Selects if the checksum is check by hardware or by software. This parameter can be a value of [ETH\\_Checksum\\_Mode](#)
- *uint32\_t ETH\_InitTypeDef::MedialInterface*
  - Selects the media-independent interface or the reduced media-independent interface. This parameter can be a value of [ETH\\_Media\\_Interface](#)

### 16.1.3 ETH\_MACInitTypeDef

*ETH\_MACInitTypeDef* is defined in the `stm32f4xx_hal_eth.h`

#### Data Fields

- *uint32\_t Watchdog*
- *uint32\_t Jabber*
- *uint32\_t InterFrameGap*
- *uint32\_t CarrierSense*
- *uint32\_t ReceiveOwn*
- *uint32\_t LoopbackMode*
- *uint32\_t ChecksumOffload*
- *uint32\_t RetryTransmission*
- *uint32\_t AutomaticPadCRCStrip*
- *uint32\_t BackOffLimit*
- *uint32\_t DeferralCheck*

- `uint32_t ReceiveAll`
- `uint32_t SourceAddrFilter`
- `uint32_t PassControlFrames`
- `uint32_t BroadcastFramesReception`
- `uint32_t DestinationAddrFilter`
- `uint32_t PromiscuousMode`
- `uint32_t MulticastFramesFilter`
- `uint32_t UnicastFramesFilter`
- `uint32_t HashTableHigh`
- `uint32_t HashTableLow`
- `uint32_t PauseTime`
- `uint32_t ZeroQuantaPause`
- `uint32_t PauseLowThreshold`
- `uint32_t UnicastPauseFrameDetect`
- `uint32_t ReceiveFlowControl`
- `uint32_t TransmitFlowControl`
- `uint32_t VLANTagComparison`
- `uint32_t VLANTagIdentifier`

### Field Documentation

- `uint32_t ETH_MACInitTypeDef::Watchdog`
  - Selects or not the Watchdog timer When enabled, the MAC allows no more than 2048 bytes to be received. When disabled, the MAC can receive up to 16384 bytes. This parameter can be a value of [ETH\\_watchdog](#)
- `uint32_t ETH_MACInitTypeDef::Jabber`
  - Selects or not Jabber timer When enabled, the MAC allows no more than 2048 bytes to be sent. When disabled, the MAC can send up to 16384 bytes. This parameter can be a value of [ETH\\_Jabber](#)
- `uint32_t ETH_MACInitTypeDef::InterFrameGap`
  - Selects the minimum IFG between frames during transmission. This parameter can be a value of [ETH\\_Inter\\_Frame\\_Gap](#)
- `uint32_t ETH_MACInitTypeDef::CarrierSense`
  - Selects or not the Carrier Sense. This parameter can be a value of [ETH\\_Carrier\\_Sense](#)
- `uint32_t ETH_MACInitTypeDef::ReceiveOwn`
  - Selects or not the ReceiveOwn, ReceiveOwn allows the reception of frames when the TX\_EN signal is asserted in Half-Duplex mode. This parameter can be a value of [ETH\\_Receive\\_Own](#)
- `uint32_t ETH_MACInitTypeDef::LoopbackMode`
  - Selects or not the internal MAC MII Loopback mode. This parameter can be a value of [ETH\\_Loop\\_Back\\_Mode](#)
- `uint32_t ETH_MACInitTypeDef::ChecksumOffload`
  - Selects or not the IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. This parameter can be a value of [ETH\\_Checksum\\_Offload](#)
- `uint32_t ETH_MACInitTypeDef::RetryTransmission`
  - Selects or not the MAC attempt retries transmission, based on the settings of BL, when a collision occurs (Half-Duplex mode). This parameter can be a value of [ETH\\_Retry\\_Transmission](#)
- `uint32_t ETH_MACInitTypeDef::AutomaticPadCRCStrip`

- Selects or not the Automatic MAC Pad/CRC Stripping. This parameter can be a value of [\*\*\*ETH\\_Automatic\\_Pad\\_CRC\\_Strip\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::BackOffLimit***
  - Selects the BackOff limit value. This parameter can be a value of [\*\*\*ETH\\_Back\\_Off\\_Limit\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::DeferralCheck***
  - Selects or not the deferral check function (Half-Duplex mode). This parameter can be a value of [\*\*\*ETH\\_Deferral\\_Check\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::ReceiveAll***
  - Selects or not all frames reception by the MAC (No filtering). This parameter can be a value of [\*\*\*ETH\\_Receive\\_All\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::SourceAddrFilter***
  - Selects the Source Address Filter mode. This parameter can be a value of [\*\*\*ETH\\_Source\\_Addr\\_Filter\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::PassControlFrames***
  - Sets the forwarding mode of the control frames (including unicast and multicast PAUSE frames) This parameter can be a value of [\*\*\*ETH\\_Pass\\_Control\\_Frames\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::BroadcastFramesReception***
  - Selects or not the reception of Broadcast Frames. This parameter can be a value of [\*\*\*ETH\\_Broadcast\\_Frames\\_Reception\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::DestinationAddrFilter***
  - Sets the destination filter mode for both unicast and multicast frames. This parameter can be a value of [\*\*\*ETH\\_Destination\\_Addr\\_Filter\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::PromiscuousMode***
  - Selects or not the Promiscuous Mode This parameter can be a value of [\*\*\*ETH\\_Promiscuous\\_Mode\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::MulticastFramesFilter***
  - Selects the Multicast Frames filter mode:  
None/HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of [\*\*\*ETH\\_Multicast\\_Frames\\_Filter\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::UnicastFramesFilter***
  - Selects the Unicast Frames filter mode:  
HashTableFilter/PerfectFilter/PerfectHashTableFilter. This parameter can be a value of [\*\*\*ETH\\_Uncast\\_Frames\\_Filter\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::HashTableHigh***
  - This field holds the higher 32 bits of Hash table. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFFFFF
- ***uint32\_t ETH\_MACInitTypeDef::HashTableLow***
  - This field holds the lower 32 bits of Hash table. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFFFFF
- ***uint32\_t ETH\_MACInitTypeDef::PauseTime***
  - This field holds the value to be used in the Pause Time field in the transmit control frame. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFF
- ***uint32\_t ETH\_MACInitTypeDef::ZeroQuantaPause***
  - Selects or not the automatic generation of Zero-Quanta Pause Control frames. This parameter can be a value of [\*\*\*ETH\\_Zero\\_Quanta\\_Pause\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::PauseLowThreshold***
  - This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Frame. This parameter can be a value of [\*\*\*ETH\\_Pause\\_Low\\_Threshold\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::UnicastPauseFrameDetect***

- Selects or not the MAC detection of the Pause frames (with MAC Address0 unicast address and unique multicast address). This parameter can be a value of [\*\*\*ETH\\_Uncast\\_Pause\\_Frame\\_Detect\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::ReceiveFlowControl***
  - Enables or disables the MAC to decode the received Pause frame and disable its transmitter for a specified time (Pause Time) This parameter can be a value of [\*\*\*ETH\\_Receive\\_Flow\\_Control\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::TransmitFlowControl***
  - Enables or disables the MAC to transmit Pause frames (Full-Duplex mode) or the MAC back-pressure operation (Half-Duplex mode) This parameter can be a value of [\*\*\*ETH\\_Transmit\\_Flow\\_Control\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::VLANTagComparison***
  - Selects the 12-bit VLAN identifier or the complete 16-bit VLAN tag for comparison and filtering. This parameter can be a value of [\*\*\*ETH\\_VLAN\\_Tag\\_Comparison\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::VLANTagIdentifier***
  - Holds the VLAN tag identifier for receive frames

#### 16.1.4 **ETH\_DMADescTypeDef**

***ETH\_DMADescTypeDef*** is defined in the `stm32f4xx_hal_eth.h`

##### Data Fields

- ***\_\_IO uint32\_t Status***
- ***uint32\_t ControlBufferSize***
- ***uint32\_t Buffer1Addr***
- ***uint32\_t Buffer2NextDescAddr***
- ***uint32\_t ExtendedStatus***
- ***uint32\_t Reserved1***
- ***uint32\_tTimeStampLow***
- ***uint32\_tTimeStampHigh***

##### Field Documentation

- ***\_\_IO uint32\_t ETH\_DMADescTypeDef::Status***
  - Status
- ***uint32\_t ETH\_DMADescTypeDef::ControlBufferSize***
  - Control and Buffer1, Buffer2 lengths
- ***uint32\_t ETH\_DMADescTypeDef::Buffer1Addr***
  - Buffer1 address pointer
- ***uint32\_t ETH\_DMADescTypeDef::Buffer2NextDescAddr***
  - Buffer2 or next descriptor address pointer Enhanced ETHERNET DMA PTP Descriptors
- ***uint32\_t ETH\_DMADescTypeDef::ExtendedStatus***
  - Extended status for PTP receive descriptor
- ***uint32\_t ETH\_DMADescTypeDef::Reserved1***
  - Reserved
- ***uint32\_t ETH\_DMADescTypeDef::TimeStampLow***
  - Time Stamp Low value for transmit and receive

- *uint32\_t ETH\_DMADescTypeDef::TimeStampHigh*
  - Time Stamp High value for transmit and receive

### 16.1.5 ETH\_DMADeInitTypeDef

*ETH\_DMADeInitTypeDef* is defined in the `stm32f4xx_hal_eth.h`

#### Data Fields

- *uint32\_t DropTCPIPChecksumErrorFrame*
- *uint32\_t ReceiveStoreForward*
- *uint32\_t FlushReceivedFrame*
- *uint32\_t TransmitStoreForward*
- *uint32\_t TransmitThresholdControl*
- *uint32\_t ForwardErrorFrames*
- *uint32\_t ForwardUndersizedGoodFrames*
- *uint32\_t ReceiveThresholdControl*
- *uint32\_t SecondFrameOperate*
- *uint32\_t AddressAlignedBeats*
- *uint32\_t FixedBurst*
- *uint32\_t RxDMA\_BurstLength*
- *uint32\_t TxDMA\_BurstLength*
- *uint32\_t EnhancedDescriptorFormat*
- *uint32\_t DescriptorSkipLength*
- *uint32\_t DMAArbitration*

#### Field Documentation

- *uint32\_t ETH\_DMADeInitTypeDef::DropTCPIPChecksumErrorFrame*
  - Selects or not the Dropping of TCP/IP Checksum Error Frames. This parameter can be a value of [\*ETH\\_Drop\\_TCP\\_IP\\_Checksum\\_Error\\_Frame\*](#)
- *uint32\_t ETH\_DMADeInitTypeDef::ReceiveStoreForward*
  - Enables or disables the Receive store and forward mode. This parameter can be a value of [\*ETH\\_Receive\\_Store\\_Forward\*](#)
- *uint32\_t ETH\_DMADeInitTypeDef::FlushReceivedFrame*
  - Enables or disables the flushing of received frames. This parameter can be a value of [\*ETH\\_Flush\\_Received\\_Frame\*](#)
- *uint32\_t ETH\_DMADeInitTypeDef::TransmitStoreForward*
  - Enables or disables Transmit store and forward mode. This parameter can be a value of [\*ETH\\_Transmit\\_Store\\_Forward\*](#)
- *uint32\_t ETH\_DMADeInitTypeDef::TransmitThresholdControl*
  - Selects or not the Transmit Threshold Control. This parameter can be a value of [\*ETH\\_Transmit\\_Threshold\\_Control\*](#)
- *uint32\_t ETH\_DMADeInitTypeDef::ForwardErrorFrames*
  - Selects or not the forward to the DMA of erroneous frames. This parameter can be a value of [\*ETH\\_Forward\\_Error\\_Frames\*](#)
- *uint32\_t ETH\_DMADeInitTypeDef::ForwardUndersizedGoodFrames*
  - Enables or disables the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC) This parameter can be a value of [\*ETH\\_Forward\\_Undersized\\_Good\\_Frames\*](#)

- *uint32\_t ETH\_DMALInitTypeDef::ReceiveThresholdControl*
  - Selects the threshold level of the Receive FIFO. This parameter can be a value of [\*ETH\\_Receive\\_Threshold\\_Control\*](#)
- *uint32\_t ETH\_DMALInitTypeDef::SecondFrameOperate*
  - Selects or not the Operate on second frame mode, which allows the DMA to process a second frame of Transmit data even before obtaining the status for the first frame. This parameter can be a value of [\*ETH\\_Second\\_Frame\\_Operate\*](#)
- *uint32\_t ETH\_DMALInitTypeDef::AddressAlignedBeats*
  - Enables or disables the Address Aligned Beats. This parameter can be a value of [\*ETH\\_Address\\_Aligned\\_Beats\*](#)
- *uint32\_t ETH\_DMALInitTypeDef::FixedBurst*
  - Enables or disables the AHB Master interface fixed burst transfers. This parameter can be a value of [\*ETH\\_Fixed\\_Burst\*](#)
- *uint32\_t ETH\_DMALInitTypeDef::RxDMABurstLength*
  - Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of [\*ETH\\_Rx\\_DMA\\_Burst\\_Length\*](#)
- *uint32\_t ETH\_DMALInitTypeDef::TxDMABurstLength*
  - Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of [\*ETH\\_Tx\\_DMA\\_Burst\\_Length\*](#)
- *uint32\_t ETH\_DMALInitTypeDef::EnhancedDescriptorFormat*
  - Enables the enhanced descriptor format. This parameter can be a value of [\*ETH\\_DMA\\_Enhanced\\_descriptor\\_format\*](#)
- *uint32\_t ETH\_DMALInitTypeDef::DescriptorSkipLength*
  - Specifies the number of word to skip between two unchained descriptors (Ring mode) This parameter must be a number between Min\_Data = 0 and Max\_Data = 32
- *uint32\_t ETH\_DMALInitTypeDef::DMAArbitration*
  - Selects the DMA Tx/Rx arbitration. This parameter can be a value of [\*ETH\\_DMA\\_Arbitration\*](#)

### 16.1.6 ETH\_DMARxFrameInfos

*ETH\_DMARxFrameInfos* is defined in the `stm32f4xx_hal_eth.h`

#### Data Fields

- *ETH\_DMADescTypeDef \* FSRxDesc*
- *ETH\_DMADescTypeDef \* LSRxDesc*
- *uint32\_t SegCount*
- *uint32\_t length*
- *uint32\_t buffer*

#### Field Documentation

- *ETH\_DMADescTypeDef\* ETH\_DMARxFrameInfos::FSRxDesc*
  - First Segment Rx Desc
- *ETH\_DMADescTypeDef\* ETH\_DMARxFrameInfos::LSRxDesc*
  - Last Segment Rx Desc
- *uint32\_t ETH\_DMARxFrameInfos::SegCount*
  - Segment count

- `uint32_t ETH_DMARxFrameInfos::length`
  - Frame length
- `uint32_t ETH_DMARxFrameInfos::buffer`
  - Frame buffer

### 16.1.7 ETH\_TypeDef

`ETH_TypeDef` is defined in the stm32f439xx.h

#### Data Fields

- `_IO uint32_t MACCR`
- `_IO uint32_t MACFFR`
- `_IO uint32_t MACHTHR`
- `_IO uint32_t MACHTLR`
- `_IO uint32_t MACMIIAR`
- `_IO uint32_t MACMIIDR`
- `_IO uint32_t MACFCR`
- `_IO uint32_t MACVLANTR`
- `uint32_t RESERVED0`
- `_IO uint32_t MACRWUFFR`
- `_IO uint32_t MACPMTCSR`
- `uint32_t RESERVED1`
- `_IO uint32_t MACSR`
- `_IO uint32_t MACIMR`
- `_IO uint32_t MACA0HR`
- `_IO uint32_t MACA0LR`
- `_IO uint32_t MACA1HR`
- `_IO uint32_t MACA1LR`
- `_IO uint32_t MACA2HR`
- `_IO uint32_t MACA2LR`
- `_IO uint32_t MACA3HR`
- `_IO uint32_t MACA3LR`
- `uint32_t RESERVED2`
- `_IO uint32_t MMCCR`
- `_IO uint32_t MMCIR`
- `_IO uint32_t MMCTIR`
- `_IO uint32_t MMCRIMR`
- `_IO uint32_t MMCTIMR`
- `uint32_t RESERVED3`
- `_IO uint32_t MMCTGFSCCR`
- `_IO uint32_t MMCTGFMSCCR`
- `uint32_t RESERVED4`
- `_IO uint32_t MMCTGFCR`
- `uint32_t RESERVED5`
- `_IO uint32_t MMCRFCECR`
- `_IO uint32_t MMCRFAECR`
- `uint32_t RESERVED6`
- `_IO uint32_t MMCRGUFCR`
- `uint32_t RESERVED7`
- `_IO uint32_t PTPTSCR`

- `__IO uint32_t PTPSSIR`
- `__IO uint32_t PTPTSHR`
- `__IO uint32_t PTPTSLR`
- `__IO uint32_t PTPTSHUR`
- `__IO uint32_t PTPTSLUR`
- `__IO uint32_t PTPTSAR`
- `__IO uint32_t PTPTTHR`
- `__IO uint32_t PTPTTLR`
- `__IO uint32_t RESERVED8`
- `__IO uint32_t PTPTSSR`
- `uint32_t RESERVED9`
- `__IO uint32_t DMABMR`
- `__IO uint32_t DMATPDR`
- `__IO uint32_t DMARPDR`
- `__IO uint32_t DMARDLAR`
- `__IO uint32_t DMATDLAR`
- `__IO uint32_t DMASR`
- `__IO uint32_t DMAOMR`
- `__IO uint32_t DMAIER`
- `__IO uint32_t DMAMFBOCR`
- `__IO uint32_t DMARSWTR`
- `uint32_t RESERVED10`
- `__IO uint32_t DMACHTDR`
- `__IO uint32_t DMACHRDR`
- `__IO uint32_t DMACHTBAR`
- `__IO uint32_t DMACHRBAR`

### Field Documentation

- `__IO uint32_t ETH_TypeDef::MACCR`
- `__IO uint32_t ETH_TypeDef::MACFFR`
- `__IO uint32_t ETH_TypeDef::MACHTHR`
- `__IO uint32_t ETH_TypeDef::MACHTLR`
- `__IO uint32_t ETH_TypeDef::MACMIIAR`
- `__IO uint32_t ETH_TypeDef::MACMIIDR`
- `__IO uint32_t ETH_TypeDef::MACFCR`
- `__IO uint32_t ETH_TypeDef::MACVLANTR`
- `uint32_t ETH_TypeDef::RESERVED0[2]`
- `__IO uint32_t ETH_TypeDef::MACRWUFFR`
- `__IO uint32_t ETH_TypeDef::MACPMTCSR`
- `uint32_t ETH_TypeDef::RESERVED1[2]`
- `__IO uint32_t ETH_TypeDef::MACSR`
- `__IO uint32_t ETH_TypeDef::MACIMR`
- `__IO uint32_t ETH_TypeDef::MACA0HR`
- `__IO uint32_t ETH_TypeDef::MACA0LR`
- `__IO uint32_t ETH_TypeDef::MACA1HR`
- `__IO uint32_t ETH_TypeDef::MACA1LR`
- `__IO uint32_t ETH_TypeDef::MACA2HR`
- `__IO uint32_t ETH_TypeDef::MACA2LR`
- `__IO uint32_t ETH_TypeDef::MACA3HR`
- `__IO uint32_t ETH_TypeDef::MACA3LR`

- `uint32_t ETH_TypeDef::RESERVED2[40]`
- `_IO uint32_t ETH_TypeDef::MMCCR`
- `_IO uint32_t ETH_TypeDef::MMCRIR`
- `_IO uint32_t ETH_TypeDef::MMCTIR`
- `_IO uint32_t ETH_TypeDef::MMCRIMR`
- `_IO uint32_t ETH_TypeDef::MMCTIMR`
- `uint32_t ETH_TypeDef::RESERVED3[14]`
- `_IO uint32_t ETH_TypeDef::MMCTGFSCCR`
- `_IO uint32_t ETH_TypeDef::MMCTGFMSCCR`
- `uint32_t ETH_TypeDef::RESERVED4[5]`
- `_IO uint32_t ETH_TypeDef::MMCTGFCCR`
- `uint32_t ETH_TypeDef::RESERVED5[10]`
- `_IO uint32_t ETH_TypeDef::MMCRFCECR`
- `_IO uint32_t ETH_TypeDef::MMCRFAECR`
- `uint32_t ETH_TypeDef::RESERVED6[10]`
- `_IO uint32_t ETH_TypeDef::MMCRGUFCR`
- `uint32_t ETH_TypeDef::RESERVED7[334]`
- `_IO uint32_t ETH_TypeDef::PTPTSCR`
- `_IO uint32_t ETH_TypeDef::PTPSSIR`
- `_IO uint32_t ETH_TypeDef::PTPTSHR`
- `_IO uint32_t ETH_TypeDef::PTPTSLR`
- `_IO uint32_t ETH_TypeDef::PTPTSHUR`
- `_IO uint32_t ETH_TypeDef::PTPTSLUR`
- `_IO uint32_t ETH_TypeDef::PTPTSAR`
- `_IO uint32_t ETH_TypeDef::PTPTTHR`
- `_IO uint32_t ETH_TypeDef::PTPTTLR`
- `_IO uint32_t ETH_TypeDef::RESERVED8`
- `_IO uint32_t ETH_TypeDef::PTPTSSR`
- `uint32_t ETH_TypeDef::RESERVED9[565]`
- `_IO uint32_t ETH_TypeDef::DMABMR`
- `_IO uint32_t ETH_TypeDef::DMATPDR`
- `_IO uint32_t ETH_TypeDef::DMARPDR`
- `_IO uint32_t ETH_TypeDef::DMARDLAR`
- `_IO uint32_t ETH_TypeDef::DMATDLAR`
- `_IO uint32_t ETH_TypeDef::DMASR`
- `_IO uint32_t ETH_TypeDef::DMAOMR`
- `_IO uint32_t ETH_TypeDef::DMAIER`
- `_IO uint32_t ETH_TypeDef::DMAMFBOCR`
- `_IO uint32_t ETH_TypeDef::DMARSWTR`
- `uint32_t ETH_TypeDef::RESERVED10[8]`
- `_IO uint32_t ETH_TypeDef::DMACHTDR`
- `_IO uint32_t ETH_TypeDef::DMACHRDR`
- `_IO uint32_t ETH_TypeDef::DMACHTBAR`
- `_IO uint32_t ETH_TypeDef::DMACHRBAR`

## 16.2 ETH Firmware driver API description

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

### 16.2.1 How to use this driver

1. Declare a ETH\_HandleTypeDef handle structure, for example: ETH\_HandleTypeDef heth;
2. Fill parameters of Init structure in heth handle
3. Call HAL\_ETH\_Init() API to initialize the Ethernet peripheral (MAC, DMA, ...)
4. Initialize the ETH low level resources through the HAL\_ETH\_MspInit() API:
  - a. Enable the Ethernet interface clock using
    - \_\_ETHMAC\_CLK\_ENABLE();
    - \_\_ETHMACTX\_CLK\_ENABLE();
    - \_\_ETHMACRX\_CLK\_ENABLE();
  - b. Initialize the related GPIO clocks
  - c. Configure Ethernet pin-out
  - d. Configure Ethernet NVIC interrupt (IT mode)
5. Initialize Ethernet DMA Descriptors in chain mode and point to allocated buffers:
  - a. HAL\_ETH\_DMATxDescListInit(); for Transmission process
  - b. HAL\_ETH\_DMARxDescListInit(); for Reception process
6. Enable MAC and DMA transmission and reception:
  - a. HAL\_ETH\_Start();
7. Prepare ETH DMA TX Descriptors and give the hand to ETH DMA to transfer the frame to MAC TX FIFO:
  - a. HAL\_ETH\_TransmitFrame();
8. Poll for a received frame in ETH RX DMA Descriptors and get received frame parameters
  - a. HAL\_ETH\_GetReceivedFrame(); (should be called into an infinite loop)
9. Get a received frame when an ETH RX interrupt occurs:
  - a. HAL\_ETH\_GetReceivedFrame\_IT(); (called in IT mode only)
10. Communicate with external PHY device:
  - a. Read a specific register from the PHY HAL\_ETH\_ReadPHYRegister();
  - b. Write data to a specific RHY register: HAL\_ETH\_WritePHYRegister();
11. Configure the Ethernet MAC after ETH peripheral initialization  
HAL\_ETH\_ConfigMAC(); all MAC parameters should be filled.
12. Configure the Ethernet DMA after ETH peripheral initialization  
HAL\_ETH\_ConfigDMA(); all DMA parameters should be filled.

### 16.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the Ethernet peripheral
- De-initialize the Ethernet peripheral
- [\*\*HAL\\_ETH\\_Init\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_Delinit\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_DMATxDescListInit\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_DMARxDescListInit\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_MspDelinit\(\)\*\*](#)

### 16.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a frame HAL\_ETH\_TransmitFrame();

- Receive a frame `HAL_ETH_GetReceivedFrame();`  
`HAL_ETH_GetReceivedFrame_IT();`
- Read from an External PHY register `HAL_ETH_ReadPHYRegister();`
- Write to an External PHY register `HAL_ETH_WritePHYRegister();`
- `HAL_ETH_TransmitFrame()`
- `HAL_ETH_GetReceivedFrame()`
- `HAL_ETH_GetReceivedFrame_IT()`
- `HAL_ETH_IRQHandler()`
- `HAL_ETH_TxCpltCallback()`
- `HAL_ETH_RxCpltCallback()`
- `HAL_ETH_ErrorCallback()`
- `HAL_ETH_ReadPHYRegister()`
- `HAL_ETH_WritePHYRegister()`

#### 16.2.4 Peripheral Control functions

This section provides functions allowing to:

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

#### 16.2.5 Peripheral State functions

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

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

#### 16.2.6 Initialization and de-initialization functions

##### 16.2.6.1 HAL\_ETH\_Init

Function Name	<code>HAL_StatusTypeDef HAL_ETH_Init ( <i>ETH_HandleTypeDef</i> * heth)</code>
Function Description	Initializes the Ethernet MAC and DMA according to default parameters.
Parameters	<ul style="list-style-type: none"> <li>• <code>heth</code> : pointer to a <i>ETH_HandleTypeDef</i> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 16.2.6.2 HAL\_ETH\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_ETH_DeInit ( <i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	De-Initializes the ETH peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a <i>ETH_HandleTypeDef</i> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 16.2.6.3 HAL\_ETH\_DMATxDescListInit

Function Name	<b>HAL_StatusTypeDef HAL_ETH_DMATxDescListInit ( <i>ETH_HandleTypeDef</i> * heth, <i>ETH_DMADescTypeDef</i> * DMATxDescTab, <i>uint8_t</i> * TxBuff, <i>uint32_t</i> TxBuffCount)</b>
Function Description	Initializes the DMA Tx descriptors in chain mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a <i>ETH_HandleTypeDef</i> structure that contains the configuration information for ETHERNET module</li> <li>• <b>DMATxDescTab</b> : Pointer to the first Tx desc list</li> <li>• <b>TxBuff</b> : Pointer to the first TxBuffer list</li> <li>• <b>TxBuffCount</b> : Number of the used Tx desc in the list</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 16.2.6.4 HAL\_ETH\_DMARxDescListInit

Function Name	<b>HAL_StatusTypeDef HAL_ETH_DMARxDescListInit ( <i>ETH_HandleTypeDef</i> * heth, <i>ETH_DMADescTypeDef</i> * DMARxDescTab, <i>uint8_t</i> * RxBuff, <i>uint32_t</i> RxBuffCount)</b>
Function Description	Initializes the DMA Rx descriptors in chain mode.

Parameters	<ul style="list-style-type: none"> <li><b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li><b>DMARxDescTab</b> : Pointer to the first Rx desc list</li> <li><b>RxBuff</b> : Pointer to the first RxBuffer list</li> <li><b>RxBuffCount</b> : Number of the used Rx desc in the list</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 16.2.6.5 HAL\_ETH\_MspInit

Function Name	<b>void HAL_ETH_MspInit ( <i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	Initializes the ETH MSP.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 16.2.6.6 HAL\_ETH\_MspDelInit

Function Name	<b>void HAL_ETH_MspDelInit ( <i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	Deinitializes ETH MSP.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 16.2.7 IO operation functions

### 16.2.7.1 HAL\_ETH\_TransmitFrame

Function Name	<b>HAL_StatusTypeDef HAL_ETH_TransmitFrame (</b> <b><i>ETH_HandleTypeDef</i> * heth, uint32_t FrameLength)</b>
Function Description	Sends an Ethernet frame.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li>• <b>FrameLength</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 16.2.7.2 HAL\_ETH\_GetReceivedFrame

Function Name	<b>HAL_StatusTypeDef HAL_ETH_GetReceivedFrame (</b> <b><i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	Checks for received frames.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 16.2.7.3 HAL\_ETH\_GetReceivedFrame\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ETH_GetReceivedFrame_IT (</b> <b><i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	Gets the Received frame in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 16.2.7.4 HAL\_ETH\_IRQHandler

Function Name	<b>void HAL_ETH_IRQHandler ( <i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	This function handles ETH interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>heth</b> : pointer to a <i>ETH_HandleTypeDef</i> structure that contains the configuration information for ETHERNET module</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 16.2.7.5 HAL\_ETH\_TxCpltCallback

Function Name	<b>void HAL_ETH_TxCpltCallback ( <i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>heth</b> : pointer to a <i>ETH_HandleTypeDef</i> structure that contains the configuration information for ETHERNET module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 16.2.7.6 HAL\_ETH\_RxCpltCallback

Function Name	<b>void HAL_ETH_RxCpltCallback ( <i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>heth</b> : pointer to a <i>ETH_HandleTypeDef</i> structure that contains the configuration information for ETHERNET module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 16.2.7.7 HAL\_ETH\_ErrorCallback

Function Name	<b>void HAL_ETH_ErrorCallback ( <i>ETH_HandleTypeDef</i> * heth)</b>
Function Description	Ethernet transfer error callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a <i>ETH_HandleTypeDef</i> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 16.2.7.8 HAL\_ETH\_ReadPHYRegister

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ReadPHYRegister ( <i>ETH_HandleTypeDef</i> * heth, uint16_t PHYReg, uint32_t * RegValue)</b>
Function Description	Reads a PHY register.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a <i>ETH_HandleTypeDef</i> structure that contains the configuration information for ETHERNET module</li> <li>• <b>PHYReg</b> : PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY_BCR: Transceiver Basic Control Register, PHY_BSR: Transceiver Basic Status Register. More PHY register could be read depending on the used PHY</li> <li>• <b>RegValue</b> : PHY register value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 16.2.7.9 HAL\_ETH\_WritePHYRegister

Function Name	<b>HAL_StatusTypeDef HAL_ETH_WritePHYRegister ( <i>ETH_HandleTypeDef</i> * heth, uint16_t PHYReg, uint32_t RegValue)</b>
Function Description	Writes to a PHY register.

Parameters	<ul style="list-style-type: none"> <li><b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li><b>PHYReg</b> : PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY_BCR : Transceiver Control Register. More More PHY register could be written depending on the used PHY</li> <li><b>RegValue</b> : the value to write</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 16.2.8 Peripheral Control functions

### 16.2.8.1 HAL\_ETH\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ETH_Start ( <i>ETH_HandleTypeDef</i> * <b>heth</b>)</b>
Function Description	Enables Ethernet MAC and DMA reception/transmission.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 16.2.8.2 HAL\_ETH\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_ETH_Stop ( <i>ETH_HandleTypeDef</i> * <b>heth</b>)</b>
Function Description	Stop Ethernet MAC and DMA reception/transmission.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 16.2.8.3 HAL\_ETH\_ConfigMAC

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ConfigMAC (</b> <i>ETH_HandleTypeDef</i> * heth, <i>ETH_MACInitTypeDef</i> * macconf)
Function Description	Set ETH MAC Configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li>• <b>macconf</b> : MAC Configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 16.2.8.4 HAL\_ETH\_ConfigDMA

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ConfigDMA (</b> <i>ETH_HandleTypeDef</i> * heth, <i>ETH_DMAMainitTypeDef</i> * dmaconf)
Function Description	Sets ETH DMA Configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li>• <b>dmaconf</b> : DMA Configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 16.2.9 Peripheral State functions

### 16.2.9.1 HAL\_ETH\_GetState

Function Name	<b>HAL_ETH_StateTypeDef HAL_ETH_GetState (</b> <i>ETH_HandleTypeDef</i> * heth)
Function Description	Return the ETH HAL state.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth</b> : pointer to a ETH_HandleTypeDef structure that</li> </ul>

---

contains the configuration information for ETHERNET module

Return values

- **HAL state**

Notes

- None.

## 16.3 ETH Firmware driver defines

### 16.3.1 ETH

ETH

*ETH\_Address\_Aligned\_Beats*

- #define: ***ETH\_ADDRESSALIGNEDBEATS\_ENABLE*** ((*uint32\_t*)0x02000000)

- #define: ***ETH\_ADDRESSALIGNEDBEATS\_DISABLE*** ((*uint32\_t*)0x00000000)

*ETH\_Automatic\_Pad\_CRC\_Strip*

- #define: ***ETH\_AUTOMATICPADCRCSTRIP\_ENABLE*** ((*uint32\_t*)0x00000080)

- #define: ***ETH\_AUTOMATICPADCRCSTRIP\_DISABLE*** ((*uint32\_t*)0x00000000)

*ETH\_AutoNegotiation*

- #define: ***ETH\_AUTONEGOTIATION\_ENABLE*** ((*uint32\_t*)0x00000001)

- #define: ***ETH\_AUTONEGOTIATION\_DISABLE*** ((*uint32\_t*)0x00000000)

*ETH\_Back\_Off\_Limit*

- #define: ***ETH\_BACKOFFLIMIT\_10*** ((*uint32\_t*)0x00000000)

- #define: ***ETH\_BACKOFFLIMIT\_8*** ((*uint32\_t*)0x00000020)

- #define: **ETH\_BACKOFFLIMIT\_4** ((*uint32\_t*)0x00000040)

- #define: **ETH\_BACKOFFLIMIT\_1** ((*uint32\_t*)0x00000060)

#### ***ETH\_Broadcast\_Frames\_Reception***

- #define: **ETH\_BROADCASTFRAMESRECEPTION\_ENABLE** ((*uint32\_t*)0x00000000)
- #define: **ETH\_BROADCASTFRAMESRECEPTION\_DISABLE** ((*uint32\_t*)0x00000020)

#### ***ETH\_Buffers\_setting***

- #define: **ETH\_MAX\_PACKET\_SIZE** ((*uint32\_t*)1524)  
*ETH\_HEADER + ETH\_EXTRA + VLAN\_TAG + MAX\_ETH\_PAYLOAD + ETH\_CRC*
- #define: **ETH\_HEADER** ((*uint32\_t*)14)  
*6 byte Dest addr, 6 byte Src addr, 2 byte length/type*
- #define: **ETH\_CRC** ((*uint32\_t*)4)  
*Ethernet CRC*
- #define: **ETH\_EXTRA** ((*uint32\_t*)2)  
*Extra bytes in some cases*
- #define: **VLAN\_TAG** ((*uint32\_t*)4)  
*optional 802.1q VLAN Tag*
- #define: **MIN\_ETH\_PAYLOAD** ((*uint32\_t*)46)  
*Minimum Ethernet payload size*
- #define: **MAX\_ETH\_PAYLOAD** ((*uint32\_t*)1500)

*Maximum Ethernet payload size*

- #define: **JUMBO\_FRAME\_PAYLOAD ((uint32\_t)9000)**  
*Jumbo frame payload size*
- #define: **ETH\_RX\_BUF\_SIZE ETH\_MAX\_PACKET\_SIZE**
- #define: **ETH\_RXBUFNB ((uint32\_t)5)**
- #define: **ETH\_TX\_BUF\_SIZE ETH\_MAX\_PACKET\_SIZE**
- #define: **ETH\_TXBUFNB ((uint32\_t)5)**
- #define: **ETH\_DMATXDESC\_OWN ((uint32\_t)0x80000000)**  
*Own bit: descriptor is owned by DMA engine*
- #define: **ETH\_DMATXDESC\_IC ((uint32\_t)0x40000000)**  
*Interrupt on Completion*
- #define: **ETH\_DMATXDESC\_LS ((uint32\_t)0x20000000)**  
*Last Segment*
- #define: **ETH\_DMATXDESC\_FS ((uint32\_t)0x10000000)**  
*First Segment*
- #define: **ETH\_DMATXDESC\_DC ((uint32\_t)0x08000000)**  
*Disable CRC*
- #define: **ETH\_DMATXDESC\_DP ((uint32\_t)0x04000000)**  
*Disable Padding*
- #define: **ETH\_DMATXDESC\_TTSE ((uint32\_t)0x02000000)**

*Transmit Time Stamp Enable*

- #define: **ETH\_DMATXDESC\_CIC** ((*uint32\_t*)0x00C00000)

*Checksum Insertion Control: 4 cases*

- #define: **ETH\_DMATXDESC\_CIC\_BYPASS** ((*uint32\_t*)0x00000000)

*Do Nothing: Checksum Engine is bypassed*

- #define: **ETH\_DMATXDESC\_CIC\_IPV4HEADER** ((*uint32\_t*)0x00400000)

*IPv4 header Checksum Insertion*

- #define: **ETH\_DMATXDESC\_CIC\_TCPUDPICMP\_SEGMENT**  
((*uint32\_t*)0x00800000)

*TCP/UDP/ICMP Checksum Insertion calculated over segment only*

- #define: **ETH\_DMATXDESC\_CIC\_TCPUDPICMP\_FULL** ((*uint32\_t*)0x00C00000)

*TCP/UDP/ICMP Checksum Insertion fully calculated*

- #define: **ETH\_DMATXDESC\_TER** ((*uint32\_t*)0x00200000)

*Transmit End of Ring*

- #define: **ETH\_DMATXDESC\_TCH** ((*uint32\_t*)0x00100000)

*Second Address Chained*

- #define: **ETH\_DMATXDESC\_TTSS** ((*uint32\_t*)0x00020000)

*Tx Time Stamp Status*

- #define: **ETH\_DMATXDESC\_IHE** ((*uint32\_t*)0x00010000)

*IP Header Error*

- #define: **ETH\_DMATXDESC\_ES** ((*uint32\_t*)0x00008000)

*Error summary: OR of the following bits: UE || ED || EC || LCO || NC || LCA || FF || JT*

- #define: **ETH\_DMATXDESC\_JT** ((*uint32\_t*)0x00004000)

*Jabber Timeout*

- #define: **ETH\_DMATXDESC\_FF** ((*uint32\_t*)0x00002000)

*Frame Flushed: DMA/MTL flushed the frame due to SW flush*

- #define: **ETH\_DMATXDESC\_PCE** ((uint32\_t)0x00001000)

*Payload Checksum Error*

- #define: **ETH\_DMATXDESC\_LCA** ((uint32\_t)0x00000800)

*Loss of Carrier: carrier lost during transmission*

- #define: **ETH\_DMATXDESC\_NC** ((uint32\_t)0x00000400)

*No Carrier: no carrier signal from the transceiver*

- #define: **ETH\_DMATXDESC\_LCO** ((uint32\_t)0x00000200)

*Late Collision: transmission aborted due to collision*

- #define: **ETH\_DMATXDESC\_EC** ((uint32\_t)0x00000100)

*Excessive Collision: transmission aborted after 16 collisions*

- #define: **ETH\_DMATXDESC\_VF** ((uint32\_t)0x00000080)

*VLAN Frame*

- #define: **ETH\_DMATXDESC\_CC** ((uint32\_t)0x00000078)

*Collision Count*

- #define: **ETH\_DMATXDESC\_ED** ((uint32\_t)0x00000004)

*Excessive Deferral*

- #define: **ETH\_DMATXDESC\_UF** ((uint32\_t)0x00000002)

*Underflow Error: late data arrival from the memory*

- #define: **ETH\_DMATXDESC\_DB** ((uint32\_t)0x00000001)

*Deferred Bit*

- #define: **ETH\_DMATXDESC\_TBS2** ((uint32\_t)0x1FFF0000)

*Transmit Buffer2 Size*

- #define: **ETH\_DMATXDESC\_TBS1** ((uint32\_t)0x00001FFF)

*Transmit Buffer1 Size*

- #define: ***ETH\_DMATXDESC\_B1AP*** ((*uint32\_t*)0xFFFFFFFF)  
*Buffer1 Address Pointer*
- #define: ***ETH\_DMATXDESC\_B2AP*** ((*uint32\_t*)0xFFFFFFFF)  
*Buffer2 Address Pointer*
- #define: ***ETH\_DMAPTPTXDESC\_TTSL*** ((*uint32\_t*)0xFFFFFFFF)
- #define: ***ETH\_DMAPTPTXDESC\_TTSH*** ((*uint32\_t*)0xFFFFFFFF)

*ETH\_CARRIER\_Sense*

- #define: ***ETH\_CARRIERSENCE\_ENABLE*** ((*uint32\_t*)0x00000000)
- #define: ***ETH\_CARRIERSENCE\_DISABLE*** ((*uint32\_t*)0x00010000)

*ETH\_Checksum\_Mode*

- #define: ***ETH\_CHECKSUM\_BY\_HARDWARE*** ((*uint32\_t*)0x00000000)
- #define: ***ETH\_CHECKSUM\_BY\_SOFTWARE*** ((*uint32\_t*)0x00000001)

*ETH\_Checksum\_Offload*

- #define: ***ETH\_CHECKSUMOFFLOAD\_ENABLE*** ((*uint32\_t*)0x00000400)
- #define: ***ETH\_CHECKSUMOFFLOAD\_DISABLE*** ((*uint32\_t*)0x00000000)

*ETH\_Deferral\_Check*

- #define: ***ETH\_DEFERRALCHECK\_ENABLE*** ((*uint32\_t*)0x00000010)
  - #define: ***ETH\_DEFERRALCHECK\_DISABLE*** ((*uint32\_t*)0x00000000)
- 
- ETH\_Destination\_Addr\_Filter***
- #define: ***ETH\_DESTINATIONADDRFILTER\_NORMAL*** ((*uint32\_t*)0x00000000)
  - #define: ***ETH\_DESTINATIONADDRFILTER\_INVERSE*** ((*uint32\_t*)0x00000008)

***ETH\_DMA\_Arbitration***

- #define: ***ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_1\_1*** ((*uint32\_t*)0x00000000)
- #define: ***ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_2\_1*** ((*uint32\_t*)0x00004000)
- #define: ***ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_3\_1*** ((*uint32\_t*)0x00008000)
- #define: ***ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_4\_1*** ((*uint32\_t*)0x0000C000)
- #define: ***ETH\_DMAARBITRATION\_RXPRIORTX*** ((*uint32\_t*)0x00000002)

***ETH\_DMA\_Enhanced\_descriptor\_format***

- #define: ***ETH\_DMAENHANCEDDESCRIPTOR\_ENABLE*** ((*uint32\_t*)0x00000080)
- #define: ***ETH\_DMAENHANCEDDESCRIPTOR\_DISABLE*** ((*uint32\_t*)0x00000000)

***ETH\_DMA\_Flags***

- #define: ***ETH\_DMA\_FLAG\_TST*** ((*uint32\_t*)0x20000000)  
*Time-stamp trigger interrupt (on DMA)*
- #define: ***ETH\_DMA\_FLAG\_PMT*** ((*uint32\_t*)0x10000000)  
*PMT interrupt (on DMA)*
- #define: ***ETH\_DMA\_FLAG\_MMC*** ((*uint32\_t*)0x08000000)  
*MMC interrupt (on DMA)*
- #define: ***ETH\_DMA\_FLAG\_DATATRANSFERERROR*** ((*uint32\_t*)0x00800000)  
*Error bits 0-Rx DMA, 1-Tx DMA*
- #define: ***ETH\_DMA\_FLAG\_READWRITEERROR*** ((*uint32\_t*)0x01000000)  
*Error bits 0-write trnsf, 1-read transfr*
- #define: ***ETH\_DMA\_FLAG\_ACCESSERROR*** ((*uint32\_t*)0x02000000)  
*Error bits 0-data buffer, 1-desc. access*
- #define: ***ETH\_DMA\_FLAG\_NIS*** ((*uint32\_t*)0x00010000)  
*Normal interrupt summary flag*
- #define: ***ETH\_DMA\_FLAG\_AIS*** ((*uint32\_t*)0x00008000)  
*Abnormal interrupt summary flag*
- #define: ***ETH\_DMA\_FLAG\_ER*** ((*uint32\_t*)0x00004000)  
*Early receive flag*
- #define: ***ETH\_DMA\_FLAG\_FBE*** ((*uint32\_t*)0x00002000)  
*Fatal bus error flag*
- #define: ***ETH\_DMA\_FLAG\_ET*** ((*uint32\_t*)0x00000400)  
*Early transmit flag*

- #define: **ETH\_DMA\_FLAG\_RWT** ((uint32\_t)0x000000200)

*Receive watchdog timeout flag*

- #define: **ETH\_DMA\_FLAG\_RPS** ((uint32\_t)0x000000100)

*Receive process stopped flag*

- #define: **ETH\_DMA\_FLAG\_RBU** ((uint32\_t)0x000000080)

*Receive buffer unavailable flag*

- #define: **ETH\_DMA\_FLAG\_R** ((uint32\_t)0x000000040)

*Receive flag*

- #define: **ETH\_DMA\_FLAG\_TU** ((uint32\_t)0x000000020)

*Underflow flag*

- #define: **ETH\_DMA\_FLAG\_RO** ((uint32\_t)0x000000010)

*Overflow flag*

- #define: **ETH\_DMA\_FLAG\_TJT** ((uint32\_t)0x000000008)

*Transmit jabber timeout flag*

- #define: **ETH\_DMA\_FLAG\_TBU** ((uint32\_t)0x000000004)

*Transmit buffer unavailable flag*

- #define: **ETH\_DMA\_FLAG\_TPS** ((uint32\_t)0x000000002)

*Transmit process stopped flag*

- #define: **ETH\_DMA\_FLAG\_T** ((uint32\_t)0x000000001)

*Transmit flag*

#### ***ETH\_DMA\_Interrupts***

- #define: **ETH\_DMA\_IT\_TST** ((uint32\_t)0x20000000)

*Time-stamp trigger interrupt (on DMA)*

- #define: **ETH\_DMA\_IT\_PMT** ((uint32\_t)0x10000000)

*PMT interrupt (on DMA)*

- #define: ***ETH\_DMA\_IT\_MMC*** ((*uint32\_t*)0x08000000)  
*MMC interrupt (on DMA)*
- #define: ***ETH\_DMA\_IT\_NIS*** ((*uint32\_t*)0x00010000)  
*Normal interrupt summary*
- #define: ***ETH\_DMA\_IT\_AIS*** ((*uint32\_t*)0x00008000)  
*Abnormal interrupt summary*
- #define: ***ETH\_DMA\_IT\_ER*** ((*uint32\_t*)0x00004000)  
*Early receive interrupt*
- #define: ***ETH\_DMA\_IT\_FBE*** ((*uint32\_t*)0x00002000)  
*Fatal bus error interrupt*
- #define: ***ETH\_DMA\_IT\_ET*** ((*uint32\_t*)0x00000400)  
*Early transmit interrupt*
- #define: ***ETH\_DMA\_IT\_RWT*** ((*uint32\_t*)0x00000200)  
*Receive watchdog timeout interrupt*
- #define: ***ETH\_DMA\_IT\_RPS*** ((*uint32\_t*)0x00000100)  
*Receive process stopped interrupt*
- #define: ***ETH\_DMA\_IT\_RBU*** ((*uint32\_t*)0x00000080)  
*Receive buffer unavailable interrupt*
- #define: ***ETH\_DMA\_IT\_R*** ((*uint32\_t*)0x00000040)  
*Receive interrupt*
- #define: ***ETH\_DMA\_IT\_TU*** ((*uint32\_t*)0x00000020)  
*Underflow interrupt*
- #define: ***ETH\_DMA\_IT\_RO*** ((*uint32\_t*)0x00000010)  
*Overflow interrupt*

- #define: **ETH\_DMA\_IT\_TJT** ((*uint32\_t*)0x00000008)

*Transmit jabber timeout interrupt*

- #define: **ETH\_DMA\_IT\_TBU** ((*uint32\_t*)0x00000004)

*Transmit buffer unavailable interrupt*

- #define: **ETH\_DMA\_IT\_TPS** ((*uint32\_t*)0x00000002)

*Transmit process stopped interrupt*

- #define: **ETH\_DMA\_IT\_T** ((*uint32\_t*)0x00000001)

*Transmit interrupt*

#### ***ETH\_DMA\_overflow\_***

- #define: **ETH\_DMA\_OVERFLOW\_RXFIFOCOUNTER** ((*uint32\_t*)0x10000000)

*Overflow bit for FIFO overflow counter*

- #define: **ETH\_DMA\_OVERFLOW\_MISSEDFRAMECOUNTER** ((*uint32\_t*)0x00010000)

*Overflow bit for missed frame counter*

#### ***ETH\_DMA\_receive\_process\_state\_***

- #define: **ETH\_DMA\_RECEIVEPROCESS\_STOPPED** ((*uint32\_t*)0x00000000)

*Stopped - Reset or Stop Rx Command issued*

- #define: **ETH\_DMA\_RECEIVEPROCESS\_FETCHING** ((*uint32\_t*)0x00020000)

*Running - fetching the Rx descriptor*

- #define: **ETH\_DMA\_RECEIVEPROCESS\_WAITING** ((*uint32\_t*)0x00060000)

*Running - waiting for packet*

- #define: **ETH\_DMA\_RECEIVEPROCESS\_SUSPENDED** ((*uint32\_t*)0x00080000)

*Suspended - Rx Descriptor unavailable*

- #define: **ETH\_DMA\_RECEIVEPROCESS\_CLOSING** ((*uint32\_t*)0x000A0000)

*Running - closing descriptor*

- #define: **ETH\_DMA\_RECEIVEPROCESS\_QUEUEING** ((*uint32\_t*)0x000E0000)

*Running - queuing the receive frame into host memory*

#### ***ETH\_DMA\_Rx\_descriptor***

- #define: **ETH\_DMARXDESC\_OWN** ((*uint32\_t*)0x80000000)

*OWN bit: descriptor is owned by DMA engine*

- #define: **ETH\_DMARXDESC\_AFM** ((*uint32\_t*)0x40000000)

*DA Filter Fail for the rx frame*

- #define: **ETH\_DMARXDESC\_FL** ((*uint32\_t*)0x3FFF0000)

*Receive descriptor frame length*

- #define: **ETH\_DMARXDESC\_ES** ((*uint32\_t*)0x00008000)

*Error summary: OR of the following bits: DE || OE || IPC || LC || RWT || RE || CE*

- #define: **ETH\_DMARXDESC\_DE** ((*uint32\_t*)0x00004000)

*Descriptor error: no more descriptors for receive frame*

- #define: **ETH\_DMARXDESC\_SAF** ((*uint32\_t*)0x00002000)

*SA Filter Fail for the received frame*

- #define: **ETH\_DMARXDESC\_LE** ((*uint32\_t*)0x00001000)

*Frame size not matching with length field*

- #define: **ETH\_DMARXDESC\_OE** ((*uint32\_t*)0x00000800)

*Overflow Error: Frame was damaged due to buffer overflow*

- #define: **ETH\_DMARXDESC\_VLAN** ((*uint32\_t*)0x00000400)

*VLAN Tag: received frame is a VLAN frame*

- #define: **ETH\_DMARXDESC\_FS** ((*uint32\_t*)0x00000200)

*First descriptor of the frame*

- #define: **ETH\_DMARXDESC\_LS** ((*uint32\_t*)0x00000100)

*Last descriptor of the frame*

- #define: **ETH\_DMARXDESC\_IPV4HCE** ((*uint32\_t*)0x00000080)

*IPC Checksum Error: Rx Ipv4 header checksum error*

- #define: **ETH\_DMARXDESC\_LC** ((*uint32\_t*)0x00000040)

*Late collision occurred during reception*

- #define: **ETH\_DMARXDESC\_FT** ((*uint32\_t*)0x00000020)

*Frame type - Ethernet, otherwise 802.3*

- #define: **ETH\_DMARXDESC\_RWT** ((*uint32\_t*)0x00000010)

*Receive Watchdog Timeout: watchdog timer expired during reception*

- #define: **ETH\_DMARXDESC\_RE** ((*uint32\_t*)0x00000008)

*Receive error: error reported by MII interface*

- #define: **ETH\_DMARXDESC\_DBE** ((*uint32\_t*)0x00000004)

*Dribble bit error: frame contains non int multiple of 8 bits*

- #define: **ETH\_DMARXDESC\_CE** ((*uint32\_t*)0x00000002)

*CRC error*

- #define: **ETH\_DMARXDESC\_MAMPCE** ((*uint32\_t*)0x00000001)

*Rx MAC Address/Payload Checksum Error: Rx MAC address matched/ Rx Payload Checksum Error*

- #define: **ETH\_DMARXDESCDIC** ((*uint32\_t*)0x80000000)

*Disable Interrupt on Completion*

- #define: **ETH\_DMARXDESC\_RBS2** ((*uint32\_t*)0x1FFF0000)

*Receive Buffer2 Size*

- #define: **ETH\_DMARXDESC\_RER** ((*uint32\_t*)0x00008000)

*Receive End of Ring*

- #define: **ETH\_DMARXDESC\_RCH** ((*uint32\_t*)0x00004000)

*Second Address Chained*

- #define: ***ETH\_DMARXDESC\_RBS1*** ((*uint32\_t*)0x00001FFF)  
*Receive Buffer1 Size*
- #define: ***ETH\_DMARXDESC\_B1AP*** ((*uint32\_t*)0xFFFFFFFF)  
*Buffer1 Address Pointer*
- #define: ***ETH\_DMARXDESC\_B2AP*** ((*uint32\_t*)0xFFFFFFFF)  
*Buffer2 Address Pointer*
- #define: ***ETH\_DMAPTPRXDESC\_PTPV*** ((*uint32\_t*)0x00002000)
- #define: ***ETH\_DMAPTPRXDESC\_PTPFT*** ((*uint32\_t*)0x00001000)
- #define: ***ETH\_DMAPTPRXDESC\_PTPMT*** ((*uint32\_t*)0x00000F00)
- #define: ***ETH\_DMAPTPRXDESC\_PTPMT\_SYNC*** ((*uint32\_t*)0x00000100)
- #define: ***ETH\_DMAPTPRXDESC\_PTPMT\_FOLLOWUP*** ((*uint32\_t*)0x00000200)
- #define: ***ETH\_DMAPTPRXDESC\_PTPMT\_DELAYREQ*** ((*uint32\_t*)0x00000300)
- #define: ***ETH\_DMAPTPRXDESC\_PTPMT\_DELAYRESP*** ((*uint32\_t*)0x00000400)
- #define: ***ETH\_DMAPTPRXDESC\_PTPMT\_PDELAYREQ\_ANNOUNCE***  
((*uint32\_t*)0x00000500)
- #define: ***ETH\_DMAPTPRXDESC\_PTPMT\_PDELAYRESP\_MANAG***  
((*uint32\_t*)0x00000600)

- #define: ***ETH\_DMAPTPRXDESC\_PTPMT\_PDELAYRESPFOLLOWUP\_SIGNAL*** ((*uint32\_t*)0x00000700)
- #define: ***ETH\_DMAPTPRXDESC\_IPV6PR*** ((*uint32\_t*)0x00000080)
- #define: ***ETH\_DMAPTPRXDESC\_IPV4PR*** ((*uint32\_t*)0x00000040)
- #define: ***ETH\_DMAPTPRXDESC\_IPCB*** ((*uint32\_t*)0x00000020)
- #define: ***ETH\_DMAPTPRXDESC\_IPPE*** ((*uint32\_t*)0x00000010)
- #define: ***ETH\_DMAPTPRXDESC\_IPHE*** ((*uint32\_t*)0x00000008)
- #define: ***ETH\_DMAPTPRXDESC\_IPPT*** ((*uint32\_t*)0x00000007)
- #define: ***ETH\_DMAPTPRXDESC\_IPPT\_UDP*** ((*uint32\_t*)0x00000001)
- #define: ***ETH\_DMAPTPRXDESC\_IPPT\_TCP*** ((*uint32\_t*)0x00000002)
- #define: ***ETH\_DMAPTPRXDESC\_IPPT\_ICMP*** ((*uint32\_t*)0x00000003)
- #define: ***ETH\_DMAPTPRXDESC\_RTSL*** ((*uint32\_t*)0xFFFFFFFF)
- #define: ***ETH\_DMAPTPRXDESC\_RTSH*** ((*uint32\_t*)0xFFFFFFFF)

***ETH\_DMA\_Rx\_descriptor\_buffers\_***

- #define: ***ETH\_DMARXDESC\_BUFFER1*** ((*uint32\_t*)0x00000000)  
*DMA Rx Desc Buffer1*
- #define: ***ETH\_DMARXDESC\_BUFFER2*** ((*uint32\_t*)0x00000001)  
*DMA Rx Desc Buffer2*
- #define: ***ETH\_DMATXDESC\_COLLISION\_COUNTSHIFT*** ((*uint32\_t*)3)
- #define: ***ETH\_DMATXDESC\_BUFFER2\_SIZESHIFT*** ((*uint32\_t*)16)
- #define: ***ETH\_DMARXDESC\_FRAME\_LENGTHSHIFT*** ((*uint32\_t*)16)
- #define: ***ETH\_DMARXDESC\_BUFFER2\_SIZESHIFT*** ((*uint32\_t*)16)
- #define: ***ETH\_DMARXDESC\_FRAMELENGTHSHIFT*** ((*uint32\_t*)16)

***ETH\_DMA\_transmit\_process\_state\_***

- #define: ***ETH\_DMA\_TRANSMITPROCESS\_STOPPED*** ((*uint32\_t*)0x00000000)  
*Stopped - Reset or Stop Tx Command issued*
- #define: ***ETH\_DMA\_TRANSMITPROCESS\_FETCHING*** ((*uint32\_t*)0x00100000)  
*Running - fetching the Tx descriptor*
- #define: ***ETH\_DMA\_TRANSMITPROCESS\_WAITING*** ((*uint32\_t*)0x00200000)  
*Running - waiting for status*
- #define: ***ETH\_DMA\_TRANSMITPROCESS\_READING*** ((*uint32\_t*)0x00300000)  
*Running - reading the data from host memory*

- #define: **ETH\_DMA\_TRANSMITPROCESS\_SUSPENDED** ((*uint32\_t*)0x00600000)  
*Suspended - Tx Descriptor unavailable*
  
- #define: **ETH\_DMA\_TRANSMITPROCESS\_CLOSING** ((*uint32\_t*)0x00700000)  
*Running - closing Rx descriptor*

#### ***ETH\_DMA\_Tx\_descriptor\_Checksum\_Insertion\_Control***

- #define: **ETH\_DMATXDESC\_CHECKSUMBYPASS** ((*uint32\_t*)0x00000000)  
*Checksum engine bypass*
  
- #define: **ETH\_DMATXDESC\_CHECKSUMIPV4HEADER** ((*uint32\_t*)0x00400000)  
*IPv4 header checksum insertion*
  
- #define: **ETH\_DMATXDESC\_CHECKSUMTCPUDPICMPSEGMENT**  
((*uint32\_t*)0x00800000)  
*TCP/UDP/ICMP checksum insertion. Pseudo header checksum is assumed to be present*
  
- #define: **ETH\_DMATXDESC\_CHECKSUMTCPUDPICMPFULL**  
((*uint32\_t*)0x00C00000)  
*TCP/UDP/ICMP checksum fully in hardware including pseudo header*

#### ***ETH\_DMA\_Tx\_descriptor\_segment***

- #define: **ETH\_DMATXDESC\_LASTSEGMENTS** ((*uint32\_t*)0x40000000)  
*Last Segment*
  
- #define: **ETH\_DMATXDESC\_FIRSTSEGMENT** ((*uint32\_t*)0x20000000)  
*First Segment*

#### ***ETH\_Drop\_TCP\_IP\_Checksum\_Error\_Frame***

- #define: **ETH\_DROPTCPIPCHKSUMERRORFRAME\_ENABLE**  
((*uint32\_t*)0x00000000)
  
- #define: **ETH\_DROPTCPIPCHKSUMERRORFRAME\_DISABLE**  
((*uint32\_t*)0x04000000)

*ETH\_Duplex\_Mode*

- #define: ***ETH\_MODE\_FULLDUPLEX*** ((*uint32\_t*)0x000000800)
  
- #define: ***ETH\_MODE\_HALFDUPLEX*** ((*uint32\_t*)0x00000000)

*ETH\_Fixed\_Burst*

- #define: ***ETH\_FIXEDBURST\_ENABLE*** ((*uint32\_t*)0x00010000)
  
- #define: ***ETH\_FIXEDBURST\_DISABLE*** ((*uint32\_t*)0x00000000)

*ETH\_Flush\_Received\_Frame*

- #define: ***ETH\_FLUSHRECEIVEDFRAME\_ENABLE*** ((*uint32\_t*)0x00000000)
  
- #define: ***ETH\_FLUSHRECEIVEDFRAME\_DISABLE*** ((*uint32\_t*)0x01000000)

*ETH\_Forward\_Error\_Frames*

- #define: ***ETH\_FORWARDERRORFRAMES\_ENABLE*** ((*uint32\_t*)0x00000080)
  
- #define: ***ETH\_FORWARDERRORFRAMES\_DISABLE*** ((*uint32\_t*)0x00000000)

*ETH\_Forward\_Undersized\_Good\_Frames*

- #define: ***ETH\_FORWARDUNDERSIZEDGOODFRAMES\_ENABLE*** ((*uint32\_t*)0x00000040)
  
- #define: ***ETH\_FORWARDUNDERSIZEDGOODFRAMES\_DISABLE*** ((*uint32\_t*)0x00000000)

***ETH\_Inter\_Frame\_Gap***

- #define: ***ETH\_INTERFRAMEGAP\_96BIT ((uint32\_t)0x00000000)***  
*minimum IFG between frames during transmission is 96Bit*
- #define: ***ETH\_INTERFRAMEGAP\_88BIT ((uint32\_t)0x00020000)***  
*minimum IFG between frames during transmission is 88Bit*
- #define: ***ETH\_INTERFRAMEGAP\_80BIT ((uint32\_t)0x00040000)***  
*minimum IFG between frames during transmission is 80Bit*
- #define: ***ETH\_INTERFRAMEGAP\_72BIT ((uint32\_t)0x00060000)***  
*minimum IFG between frames during transmission is 72Bit*
- #define: ***ETH\_INTERFRAMEGAP\_64BIT ((uint32\_t)0x00080000)***  
*minimum IFG between frames during transmission is 64Bit*
- #define: ***ETH\_INTERFRAMEGAP\_56BIT ((uint32\_t)0x000A0000)***  
*minimum IFG between frames during transmission is 56Bit*
- #define: ***ETH\_INTERFRAMEGAP\_48BIT ((uint32\_t)0x000C0000)***  
*minimum IFG between frames during transmission is 48Bit*
- #define: ***ETH\_INTERFRAMEGAP\_40BIT ((uint32\_t)0x000E0000)***  
*minimum IFG between frames during transmission is 40Bit*

***ETH\_Jabber***

- #define: ***ETH\_JABBER\_ENABLE ((uint32\_t)0x00000000)***
- #define: ***ETH\_JABBER\_DISABLE ((uint32\_t)0x00400000)***

***ETH\_Loop\_Back\_Mode***

- #define: ***ETH\_LOOPBACKMODE\_ENABLE ((uint32\_t)0x00001000)***

- #define: **ETH\_LOOPBACKMODE\_DISABLE** ((*uint32\_t*)0x00000000)

***ETH\_MAC\_addresses***

- #define: **ETH\_MAC\_ADDRESS0** ((*uint32\_t*)0x00000000)
- #define: **ETH\_MAC\_ADDRESS1** ((*uint32\_t*)0x00000008)
- #define: **ETH\_MAC\_ADDRESS2** ((*uint32\_t*)0x00000010)
- #define: **ETH\_MAC\_ADDRESS3** ((*uint32\_t*)0x00000018)

***ETH\_MAC\_addresses\_filter\_Mask\_bytes***

- #define: **ETH\_MAC\_ADDRESSMASK\_BYTE6** ((*uint32\_t*)0x20000000)  
Mask MAC Address high reg bits [15:8]
- #define: **ETH\_MAC\_ADDRESSMASK\_BYTE5** ((*uint32\_t*)0x10000000)  
Mask MAC Address high reg bits [7:0]
- #define: **ETH\_MAC\_ADDRESSMASK\_BYTE4** ((*uint32\_t*)0x08000000)  
Mask MAC Address low reg bits [31:24]
- #define: **ETH\_MAC\_ADDRESSMASK\_BYTE3** ((*uint32\_t*)0x04000000)  
Mask MAC Address low reg bits [23:16]
- #define: **ETH\_MAC\_ADDRESSMASK\_BYTE2** ((*uint32\_t*)0x02000000)  
Mask MAC Address low reg bits [15:8]
- #define: **ETH\_MAC\_ADDRESSMASK\_BYTE1** ((*uint32\_t*)0x01000000)  
Mask MAC Address low reg bits [7:0]

***ETH\_MAC\_addresses\_filter\_SA\_DA\_filed\_of\_received\_frames***

- #define: ***ETH\_MAC\_ADDRESSFILTER\_SA*** ((*uint32\_t*)0x00000000)

- #define: ***ETH\_MAC\_ADDRESSFILTER\_DA*** ((*uint32\_t*)0x00000008)

***ETH\_MAC\_Debug\_flags***

- #define: ***ETH\_MAC\_TXFIFO\_FULL*** ((*uint32\_t*)0x02000000)

- #define: ***ETH\_MAC\_TXFIFONOT\_EMPTY*** ((*uint32\_t*)0x01000000)

- #define: ***ETH\_MAC\_TXFIFO\_WRITE\_ACTIVE*** ((*uint32\_t*)0x00400000)

- #define: ***ETH\_MAC\_TXFIFO\_IDLE*** ((*uint32\_t*)0x00000000)

- #define: ***ETH\_MAC\_TXFIFO\_READ*** ((*uint32\_t*)0x00100000)

- #define: ***ETH\_MAC\_TXFIFO\_WAITING*** ((*uint32\_t*)0x00200000)

- #define: ***ETH\_MAC\_TXFIFO\_WRITING*** ((*uint32\_t*)0x00300000)

- #define: ***ETH\_MAC\_TRANSMISSION\_PAUSE*** ((*uint32\_t*)0x00080000)

- #define: ***ETH\_MAC\_TRANSMITFRAMECONTROLLER\_IDLE***  
((*uint32\_t*)0x00000000)

- #define: ***ETH\_MAC\_TRANSMITFRAMECONTROLLER\_WAITING***  
((*uint32\_t*)0x00020000)

- #define: ***ETH\_MAC\_TRANSMITFRAMECONTROLLER\_GENRATING\_PCF***  
((uint32\_t)0x00040000)
- #define: ***ETH\_MAC\_TRANSMITFRAMECONTROLLER\_TRANSFERRING***  
((uint32\_t)0x00060000)
- #define: ***ETH\_MAC\_MII\_TRANSMIT\_ACTIVE*** ((uint32\_t)0x00010000)
- #define: ***ETH\_MAC\_RXFIFO\_EMPTY*** ((uint32\_t)0x00000000)
- #define: ***ETH\_MAC\_RXFIFO\_BELOW\_THRESHOLD*** ((uint32\_t)0x00000100)
- #define: ***ETH\_MAC\_RXFIFO\_ABOVE\_THRESHOLD*** ((uint32\_t)0x00000200)
- #define: ***ETH\_MAC\_RXFIFO\_FULL*** ((uint32\_t)0x00000300)
- #define: ***ETH\_MAC\_READCONTROLLER\_IDLE*** ((uint32\_t)0x00000060)
- #define: ***ETH\_MAC\_READCONTROLLER\_READING\_DATA***  
((uint32\_t)0x00000060)
- #define: ***ETH\_MAC\_READCONTROLLER\_READING\_STATUS***  
((uint32\_t)0x00000060)
- #define: ***ETH\_MAC\_READCONTROLLER\_FLUSHING*** ((uint32\_t)0x00000060)

- #define: ***ETH\_MAC\_RXFIFO\_WRITE\_ACTIVE*** ((*uint32\_t*)0x00000010)
- #define: ***ETH\_MAC\_SMALL\_FIFO\_NOTACTIVE*** ((*uint32\_t*)0x00000000)
- #define: ***ETH\_MAC\_SMALL\_FIFO\_READ\_ACTIVE*** ((*uint32\_t*)0x00000002)
- #define: ***ETH\_MAC\_SMALL\_FIFO\_WRITE\_ACTIVE*** ((*uint32\_t*)0x00000004)
- #define: ***ETH\_MAC\_SMALL\_FIFO\_RW\_ACTIVE*** ((*uint32\_t*)0x00000006)
- #define: ***ETH\_MAC\_MII\_RECEIVE\_PROTOCOL\_AVTIVE*** ((*uint32\_t*)0x00000001)

#### ***ETH\_MAC\_Flags***

- #define: ***ETH\_MAC\_FLAG\_TST*** ((*uint32\_t*)0x00000200)  
*Time stamp trigger flag (on MAC)*
- #define: ***ETH\_MAC\_FLAG\_MMCT*** ((*uint32\_t*)0x00000040)  
*MMC transmit flag*
- #define: ***ETH\_MAC\_FLAG\_MMCR*** ((*uint32\_t*)0x00000020)  
*MMC receive flag*
- #define: ***ETH\_MAC\_FLAG\_MMIC*** ((*uint32\_t*)0x00000010)  
*MMC flag (on MAC)*
- #define: ***ETH\_MAC\_FLAG\_PMT*** ((*uint32\_t*)0x00000008)  
*PMT flag (on MAC)*

#### ***ETH\_MAC\_Interrupts***

- #define: ***ETH\_MAC\_IT\_TST ((uint32\_t)0x00000200)***  
*Time stamp trigger interrupt (on MAC)*
- #define: ***ETH\_MAC\_IT\_MMCT ((uint32\_t)0x00000040)***  
*MMC transmit interrupt*
- #define: ***ETH\_MAC\_IT\_MMCR ((uint32\_t)0x00000020)***  
*MMC receive interrupt*
- #define: ***ETH\_MAC\_IT\_MMCI ((uint32\_t)0x00000010)***  
*MMC interrupt (on MAC)*
- #define: ***ETH\_MAC\_IT\_PMT ((uint32\_t)0x00000008)***  
*PMT interrupt (on MAC)*

#### ***ETH\_Media\_Interface***

- #define: ***ETH\_MEDIA\_INTERFACE\_MII ((uint32\_t)0x00000000)***
- #define: ***ETH\_MEDIA\_INTERFACE\_RMII ((uint32\_t)SYSCFG\_PMC\_MII\_RMII\_SEL)***

#### ***ETH\_MMC\_Registers***

- #define: ***ETH\_MMCCR ((uint32\_t)0x00000100)***  
*MMC CR register*
- #define: ***ETH\_MMCRIR ((uint32\_t)0x00000104)***  
*MMC RIR register*
- #define: ***ETH\_MMCTIR ((uint32\_t)0x00000108)***  
*MMC TIR register*
- #define: ***ETH\_MMCRIMR ((uint32\_t)0x0000010C)***  
*MMC RIMR register*
- #define: ***ETH\_MMCTIMR ((uint32\_t)0x00000110)***

*MMC TIMR register*

- #define: ***ETH\_MMCTGFSCCR ((uint32\_t)0x0000014C)***

*MMC TGFSCCR register*

- #define: ***ETH\_MMCTGFMSCCR ((uint32\_t)0x00000150)***

*MMC TGFMSCCR register*

- #define: ***ETH\_MMCTGFCR ((uint32\_t)0x00000168)***

*MMC TGFCR register*

- #define: ***ETH\_MMCRFCECR ((uint32\_t)0x00000194)***

*MMC RFCECR register*

- #define: ***ETH\_MMCRFAECR ((uint32\_t)0x00000198)***

*MMC RFAECR register*

- #define: ***ETH\_MMCRGUFCR ((uint32\_t)0x000001C4)***

*MMC RGUFCR register*

***ETH\_MMC\_Rx\_Interrupts***

- #define: ***ETH\_MMC\_IT\_RGUF ((uint32\_t)0x10020000)***

*When Rx good unicast frames counter reaches half the maximum value*

- #define: ***ETH\_MMC\_IT\_RFAE ((uint32\_t)0x10000040)***

*When Rx alignment error counter reaches half the maximum value*

- #define: ***ETH\_MMC\_IT\_RFCE ((uint32\_t)0x10000020)***

*When Rx crc error counter reaches half the maximum value*

***ETH\_MMC\_Tx\_Interrupts***

- #define: ***ETH\_MMC\_IT\_TGF ((uint32\_t)0x00200000)***

*When Tx good frame counter reaches half the maximum value*

- #define: ***ETH\_MMC\_IT\_TGFMSC ((uint32\_t)0x00008000)***

*When Tx good multi col counter reaches half the maximum value*

- #define: **ETH\_MMC\_IT\_TGFSC** ((*uint32\_t*)0x00004000)

*When Tx good single col counter reaches half the maximum value*

#### ***ETH\_Multicast\_Frames\_Filter***

- #define: **ETH\_MULTICASTFRAMESFILTER\_PERFECTHASHTABLE** ((*uint32\_t*)0x00000404)

- #define: **ETH\_MULTICASTFRAMESFILTER\_HASHTABLE** ((*uint32\_t*)0x00000004)

- #define: **ETH\_MULTICASTFRAMESFILTER\_PERFECT** ((*uint32\_t*)0x00000000)

- #define: **ETH\_MULTICASTFRAMESFILTER\_NONE** ((*uint32\_t*)0x00000010)

#### ***ETH\_Pass\_Control\_Frames***

- #define: **ETH\_PASSCONTROLFRAMES\_BLOCKALL** ((*uint32\_t*)0x00000040)

*MAC filters all control frames from reaching the application*

- #define: **ETH\_PASSCONTROLFRAMES\_FORWARDALL** ((*uint32\_t*)0x00000080)

*MAC forwards all control frames to application even if they fail the Address Filter*

- #define: **ETH\_PASSCONTROLFRAMES\_FORWARDPASSEDADDRFILTER** ((*uint32\_t*)0x000000C0)

*MAC forwards control frames that pass the Address Filter.*

#### ***ETH\_Pause\_Low\_Threshold***

- #define: **ETH\_PAUSELOWTHRESHOLD\_MINUS4** ((*uint32\_t*)0x00000000)

*Pause time minus 4 slot times*

- #define: **ETH\_PAUSELOWTHRESHOLD\_MINUS28** ((*uint32\_t*)0x00000010)

*Pause time minus 28 slot times*

- #define: **ETH\_PAUSELOWTHRESHOLD\_MINUS144** ((*uint32\_t*)0x00000020)

*Pause time minus 144 slot times*

- #define: ***ETH\_PAUSELOWTHRESHOLD\_MINUS256*** ((*uint32\_t*)0x00000030)

*Pause time minus 256 slot times*

#### ***ETH\_PMT\_Flags***

- #define: ***ETH\_PMT\_FLAG\_WUFRPR*** ((*uint32\_t*)0x80000000)

*Wake-Up Frame Filter Register Pointer Reset*

- #define: ***ETH\_PMT\_FLAG\_WUFR*** ((*uint32\_t*)0x00000040)

*Wake-Up Frame Received*

- #define: ***ETH\_PMT\_FLAG\_MPR*** ((*uint32\_t*)0x00000020)

*Magic Packet Received*

#### ***ETH\_Promiscuous\_Mode***

- #define: ***ETH\_PROMISCIOUSMODE\_ENABLE*** ((*uint32\_t*)0x00000001)

- #define: ***ETH\_PROMISCIOUSMODE\_DISABLE*** ((*uint32\_t*)0x00000000)

#### ***ETH\_Receive\_All***

- #define: ***ETH\_RECEIVEALL\_ENABLE*** ((*uint32\_t*)0x80000000)

- #define: ***ETH\_RECEIVEALL\_DISABLE*** ((*uint32\_t*)0x00000000)

#### ***ETH\_Receive\_Flow\_Control***

- #define: ***ETH\_RECEIVEFLOWCONTROL\_ENABLE*** ((*uint32\_t*)0x00000004)

- #define: ***ETH\_RECEIVEFLOWCONTROL\_DISABLE*** ((*uint32\_t*)0x00000000)

***ETH\_Receive\_Own***

- #define: ***ETH\_RECEIVEOWN\_ENABLE*** ((*uint32\_t*)0x00000000)

- #define: ***ETH\_RECEIVEOWN\_DISABLE*** ((*uint32\_t*)0x00002000)

***ETH\_Receive\_Store\_Forward***

- #define: ***ETH\_RECEIVESTOREFORWARD\_ENABLE*** ((*uint32\_t*)0x02000000)

- #define: ***ETH\_RECEIVESTOREFORWARD\_DISABLE*** ((*uint32\_t*)0x00000000)

***ETH\_Receive\_Threshold\_Control***

- #define: ***ETH\_RECEIVEDTHRESHOLDCONTROL\_64BYTES***  
((*uint32\_t*)0x00000000)

*threshold level of the MTL Receive FIFO is 64 Bytes*

- #define: ***ETH\_RECEIVEDTHRESHOLDCONTROL\_32BYTES***  
((*uint32\_t*)0x00000008)

*threshold level of the MTL Receive FIFO is 32 Bytes*

- #define: ***ETH\_RECEIVEDTHRESHOLDCONTROL\_96BYTES***  
((*uint32\_t*)0x00000010)

*threshold level of the MTL Receive FIFO is 96 Bytes*

- #define: ***ETH\_RECEIVEDTHRESHOLDCONTROL\_128BYTES***  
((*uint32\_t*)0x00000018)

*threshold level of the MTL Receive FIFO is 128 Bytes*

***ETH\_Retry\_Transmission***

- #define: ***ETH\_RETRYTRANSMISSION\_ENABLE*** ((*uint32\_t*)0x00000000)

- #define: ***ETH\_RETRYTRANSMISSION\_DISABLE*** ((*uint32\_t*)0x00000200)

***ETH\_Rx\_DMA\_Burst\_Length***

- #define: ***ETH\_RXDMABURSTLENGTH\_1BEAT ((uint32\_t)0x00020000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 1*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_2BEAT ((uint32\_t)0x00040000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 2*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_4BEAT ((uint32\_t)0x00080000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 4*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_8BEAT ((uint32\_t)0x00100000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 8*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_16BEAT ((uint32\_t)0x00200000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 16*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_32BEAT ((uint32\_t)0x00400000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 32*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_4XPBL\_4BEAT ((uint32\_t)0x01020000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 4*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_4XPBL\_8BEAT ((uint32\_t)0x01040000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 8*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_4XPBL\_16BEAT ((uint32\_t)0x01080000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 16*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_4XPBL\_32BEAT ((uint32\_t)0x01100000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 32*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_4XPBL\_64BEAT ((uint32\_t)0x01200000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 64*
  
- #define: ***ETH\_RXDMABURSTLENGTH\_4XPBL\_128BEAT ((uint32\_t)0x01400000)***  
*maximum number of beats to be transferred in one RxDMA transaction is 128*

***ETH\_Rx\_Mode***

- #define: ***ETH\_RXPOLLING\_MODE*** ((*uint32\_t*)0x00000000)
- #define: ***ETH\_RXINTERRUPT\_MODE*** ((*uint32\_t*)0x00000001)

***ETH\_Second\_Frame\_Operate***

- #define: ***ETH\_SECONDFRAMEOPERARTE\_ENABLE*** ((*uint32\_t*)0x00000004)
- #define: ***ETH\_SECONDFRAMEOPERARTE\_DISABLE*** ((*uint32\_t*)0x00000000)

***ETH\_Source\_Addr\_Filter***

- #define: ***ETH\_SOURCEADDRFILTER\_NORMAL\_ENABLE*** ((*uint32\_t*)0x00000200)
- #define: ***ETH\_SOURCEADDRFILTER\_INVERSE\_ENABLE*** ((*uint32\_t*)0x00000300)
- #define: ***ETH\_SOURCEADDRFILTER\_DISABLE*** ((*uint32\_t*)0x00000000)

***ETH\_Speed***

- #define: ***ETH\_SPEED\_10M*** ((*uint32\_t*)0x00000000)
- #define: ***ETH\_SPEED\_100M*** ((*uint32\_t*)0x00004000)

***ETH\_Transmit\_Flow\_Control***

- #define: ***ETH\_TRANSMITFLOWCONTROL\_ENABLE*** ((*uint32\_t*)0x00000002)

- #define: ***ETH\_TRANSMITFLOWCONTROL\_DISABLE*** ((*uint32\_t*)0x00000000)

#### ***ETH\_Transmit\_Store\_Foward***

- #define: ***ETH\_TRANSMITSTOREFORWARD\_ENABLE*** ((*uint32\_t*)0x00200000)
- #define: ***ETH\_TRANSMITSTOREFORWARD\_DISABLE*** ((*uint32\_t*)0x00000000)

#### ***ETH\_Transmit\_Threshold\_Control***

- #define: ***ETH\_TRANSMITTHRESHOLDCONTROL\_64BYTES***  
((*uint32\_t*)0x00000000)  
*threshold level of the MTL Transmit FIFO is 64 Bytes*
- #define: ***ETH\_TRANSMITTHRESHOLDCONTROL\_128BYTES***  
((*uint32\_t*)0x00004000)  
*threshold level of the MTL Transmit FIFO is 128 Bytes*

- #define: ***ETH\_TRANSMITTHRESHOLDCONTROL\_192BYTES***  
((*uint32\_t*)0x00008000)  
*threshold level of the MTL Transmit FIFO is 192 Bytes*
- #define: ***ETH\_TRANSMITTHRESHOLDCONTROL\_256BYTES***  
((*uint32\_t*)0x0000C000)  
*threshold level of the MTL Transmit FIFO is 256 Bytes*

- #define: ***ETH\_TRANSMITTHRESHOLDCONTROL\_40BYTES***  
((*uint32\_t*)0x00010000)  
*threshold level of the MTL Transmit FIFO is 40 Bytes*
- #define: ***ETH\_TRANSMITTHRESHOLDCONTROL\_32BYTES***  
((*uint32\_t*)0x00014000)  
*threshold level of the MTL Transmit FIFO is 32 Bytes*

- #define: ***ETH\_TRANSMITTHRESHOLDCONTROL\_24BYTES***  
((*uint32\_t*)0x00018000)  
*threshold level of the MTL Transmit FIFO is 24 Bytes*

- #define: **ETH\_TRANSMITTHRESHOLDCONTROL\_16BYTES**  
((uint32\_t)0x0001C000)

*threshold level of the MTL Transmit FIFO is 16 Bytes*

#### **ETH\_Tx\_DMA\_Burst\_Length**

- #define: **ETH\_TXDMABURSTLENGTH\_1BEAT** ((uint32\_t)0x00000100)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 1*

- #define: **ETH\_TXDMABURSTLENGTH\_2BEAT** ((uint32\_t)0x00000200)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 2*

- #define: **ETH\_TXDMABURSTLENGTH\_4BEAT** ((uint32\_t)0x00000400)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 4*

- #define: **ETH\_TXDMABURSTLENGTH\_8BEAT** ((uint32\_t)0x00000800)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 8*

- #define: **ETH\_TXDMABURSTLENGTH\_16BEAT** ((uint32\_t)0x00001000)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 16*

- #define: **ETH\_TXDMABURSTLENGTH\_32BEAT** ((uint32\_t)0x00002000)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 32*

- #define: **ETH\_TXDMABURSTLENGTH\_4XPBL\_4BEAT** ((uint32\_t)0x01000100)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 4*

- #define: **ETH\_TXDMABURSTLENGTH\_4XPBL\_8BEAT** ((uint32\_t)0x01000200)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 8*

- #define: **ETH\_TXDMABURSTLENGTH\_4XPBL\_16BEAT** ((uint32\_t)0x01000400)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 16*

- #define: **ETH\_TXDMABURSTLENGTH\_4XPBL\_32BEAT** ((uint32\_t)0x01000800)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 32*

- #define: **ETH\_TXDMABURSTLENGTH\_4XPBL\_64BEAT** ((uint32\_t)0x01001000)

*maximum number of beats to be transferred in one TxDMA (or both) transaction is 64*

- #define: ***ETH\_TXDMABURSTLENGTH\_4XPBL\_128BEAT*** ((*uint32\_t*)0x01002000)  
*maximum number of beats to be transferred in one TxDMA (or both) transaction is 128*
- #define: ***ETH\_MAC\_ADDR\_HBASE*** (*uint32\_t*)(***ETH\_MAC\_BASE*** + (*uint32\_t*)0x40)
- #define: ***ETH\_MAC\_ADDR\_LBASE*** (*uint32\_t*)(***ETH\_MAC\_BASE*** + (*uint32\_t*)0x44)
- #define: ***MACMIIAR\_CR\_MASK*** ((*uint32\_t*)0xFFFFFE3)
- #define: ***MACCR\_CLEAR\_MASK*** ((*uint32\_t*)0xFF20810F)
- #define: ***MACFCR\_CLEAR\_MASK*** ((*uint32\_t*)0x0000FF41)
- #define: ***DMAOMR\_CLEAR\_MASK*** ((*uint32\_t*)0xF8DE3F23)
- #define: ***ETH\_WAKEUP\_REGISTER\_LENGTH*** 8
- #define: ***ETH\_DMA\_RX\_OVERFLOW\_MISSEDFRAMES\_COUNTERSHIFT*** 17

#### ***ETH\_Uncast\_Frames\_Filter***

- #define: ***ETH\_UNICASTFRAMESFILTER\_PERFECTHASHTABLE*** ((*uint32\_t*)0x00000402)
- #define: ***ETH\_UNICASTFRAMESFILTER\_HASHTABLE*** ((*uint32\_t*)0x00000002)
- #define: ***ETH\_UNICASTFRAMESFILTER\_PERFECT*** ((*uint32\_t*)0x00000000)

***ETH\_Uncast\_Pause\_Frame\_Detect***

- #define: ***ETH\_UNICASTPAUSEFRAMEDETECT\_ENABLE ((uint32\_t)0x00000008)***
  
- #define: ***ETH\_UNICASTPAUSEFRAMEDETECT\_DISABLE ((uint32\_t)0x00000000)***

***ETH\_VLAN\_Tag\_Comparison***

- #define: ***ETH\_VLANTAGCOMPARISON\_12BIT ((uint32\_t)0x00010000)***
  
- #define: ***ETH\_VLANTAGCOMPARISON\_16BIT ((uint32\_t)0x00000000)***

***ETH\_watchdog***

- #define: ***ETH\_WATCHDOG\_ENABLE ((uint32\_t)0x00000000)***
  
- #define: ***ETH\_WATCHDOG\_DISABLE ((uint32\_t)0x00800000)***

***ETH\_Zero\_Quanta\_Pause***

- #define: ***ETH\_ZEROQUANTAPAUSE\_ENABLE ((uint32\_t)0x00000000)***
  
- #define: ***ETH\_ZEROQUANTAPAUSE\_DISABLE ((uint32\_t)0x00000080)***

## 17 HAL FLASH Generic Driver

### 17.1 FLASH Firmware driver registers structures

#### 17.1.1 FLASH\_ProcessTypeDef

*FLASH\_ProcessTypeDef* is defined in the `stm32f4xx_hal_flash.h`

##### Data Fields

- `__IO FLASH_ProcedureTypeDef ProcedureOnGoing`
- `__IO uint32_t NbSectorsToErase`
- `__IO uint8_t VoltageForErase`
- `__IO uint32_t Sector`
- `__IO uint32_t Bank`
- `__IO uint32_t Address`
- `HAL_LockTypeDef Lock`
- `__IO FLASH_ErrorTypeDef ErrorCode`

##### Field Documentation

- `__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing`
- `__IO uint32_t FLASH_ProcessTypeDef::NbSectorsToErase`
- `__IO uint8_t FLASH_ProcessTypeDef::VoltageForErase`
- `__IO uint32_t FLASH_ProcessTypeDef::Sector`
- `__IO uint32_t FLASH_ProcessTypeDef::Bank`
- `__IO uint32_t FLASH_ProcessTypeDef::Address`
- `HAL_LockTypeDef FLASH_ProcessTypeDef::Lock`
- `__IO FLASH_ErrorTypeDef FLASH_ProcessTypeDef::ErrorCode`

#### 17.1.2 FLASH\_TypeDef

*FLASH\_TypeDef* is defined in the `stm32f439xx.h`

##### Data Fields

- `__IO uint32_t ACR`
- `__IO uint32_t KEYR`
- `__IO uint32_t OPTKEYR`
- `__IO uint32_t SR`
- `__IO uint32_t CR`
- `__IO uint32_t OPTCR`
- `__IO uint32_t OPTCR1`

##### Field Documentation

- **`_IO uint32_t FLASH_TypeDef::ACR`**
  - FLASH access control register, Address offset: 0x00
- **`_IO uint32_t FLASH_TypeDef::KEYR`**
  - FLASH key register, Address offset: 0x04
- **`_IO uint32_t FLASH_TypeDef::OPTKEYR`**
  - FLASH option key register, Address offset: 0x08
- **`_IO uint32_t FLASH_TypeDef::SR`**
  - FLASH status register, Address offset: 0x0C
- **`_IO uint32_t FLASH_TypeDef::CR`**
  - FLASH control register, Address offset: 0x10
- **`_IO uint32_t FLASH_TypeDef::OPTCR`**
  - FLASH option control register , Address offset: 0x14
- **`_IO uint32_t FLASH_TypeDef::OPTCR1`**
  - FLASH option control register 1, Address offset: 0x18

## 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 and cache lines.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Prefetch on I-Code
- 64 cache lines of 128 bits on I-Code
- 8 cache lines of 128 bits on D-Code

### 17.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32F4xx devices.

1. FLASH Memory IO Programming functions:
  - Lock and Unlock the FLASH interface using `HAL_FLASH_Unlock()` and `HAL_FLASH_Lock()` functions
  - Program functions: byte, half word, word and double word
  - There Two modes of programming :
    - Polling mode using `HAL_FLASH_Program()` function
    - Interrupt mode using `HAL_FLASH_Program_IT()` function
2. Interrupts and flags management functions :
  - handle FLASH interrupts by calling `HAL_FLASH_IRQHandler()`
  - Wait for last FLASH operation according to its status

- Get error flag status by calling HAL\_SetErrorCode()

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

- Set the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the Instruction cache and the Data cache
- Reset the Instruction cache and the Data cache
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

### 17.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

- [\*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 Errors functions

This subsection permits to get in run-time Errors of the FLASH peripheral.

- [\*HAL\\_FLASH\\_GetError\(\)\*](#)

### 17.2.6 Programming operation functions

#### 17.2.6.1 HAL\_FLASH\_Program

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Program ( uint32_t TypeProgram, uint32_t Address, uint64_t Data)</b>
Function Description	Program byte, halfword, word or double word at a specified address.
Parameters	<ul style="list-style-type: none"> <li>• <b>TypeProgram</b> : Indicate the way to program at a specified address. This parameter can be a value of FLASH Type</li> </ul>

	Program
	<ul style="list-style-type: none"><li>• <b>Address</b> : specifies the address to be programmed.</li></ul>
	<ul style="list-style-type: none"><li>• <b>Data</b> : specifies the data to be programmed</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 17.2.6.2 HAL\_FLASH\_Program\_IT

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Program_IT ( uint32_t TypeProgram, uint32_t Address, uint64_t Data)</b>
Function Description	Program byte, halfword, word or double word at a specified address with interrupt enabled.
Parameters	<ul style="list-style-type: none"><li>• <b>TypeProgram</b> : Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program</li><li>• <b>Address</b> : specifies the address to be programmed.</li><li>• <b>Data</b> : specifies the data to be programmed</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 17.2.6.3 HAL\_FLASH\_IRQHandler

Function Name	<b>void HAL_FLASH_IRQHandler ( void )</b>
Function Description	This function handles FLASH interrupt request.
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 17.2.6.4 HAL\_FLASH\_EndOfOperationCallback

Function Name	<b>void HAL_FLASH_EndOfOperationCallback ( uint32_t ReturnValue)</b>
Function Description	FLASH end of operation interrupt callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>ReturnValue</b> : The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Sectors Erase: Sector which has been erased (if 0xFFFFFFFF, it means that all the selected sectors have been erased) Program: Address which was selected for data program</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 17.2.6.5 HAL\_FLASH\_OperationErrorCallback

Function Name	<b>void HAL_FLASH_OperationErrorCallback ( uint32_t ReturnValue)</b>
Function Description	FLASH operation error interrupt callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>ReturnValue</b> : The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Sectors Erase: Sector number which returned an error Program: Address which was selected for data program</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 17.2.7 Peripheral Control functions

### 17.2.7.1 HAL\_FLASH\_Unlock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Unlock ( void )</b>
Function Description	Unlock the FLASH control register access.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL Status</b></li> </ul>

## Notes

- None.

### 17.2.7.2 HAL\_FLASH\_Lock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Lock ( void )</b>
Function Description	Locks the FLASH control register access.
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 17.2.7.3 HAL\_FLASH\_OB\_Unlock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Unlock ( void )</b>
Function Description	Unlock the FLASH Option Control Registers access.
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 17.2.7.4 HAL\_FLASH\_OB\_Lock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Lock ( void )</b>
Function Description	Lock the FLASH Option Control Registers access.
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 17.2.7.5 HAL\_FLASH\_OB\_Launch

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Launch ( void )</b>
Function Description	Launch the option byte loading.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>HAL Status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 17.2.8 Peripheral State and Errors functions

#### 17.2.8.1 HAL\_FLASH\_GetError

Function Name	<b>FLASH_ErrorTypeDef HAL_FLASH_GetError ( void )</b>
Function Description	Get the specific FLASH error flag.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>FLASH_ErrorCode</b> : The returned value can be:<ul style="list-style-type: none"><li><b>FLASH_ERROR_RD</b>: <i>FLASH Read Protection error flag (PCROP)</i></li><li><b>FLASH_ERROR_PGS</b>: <i>FLASH Programming Sequence error flag</i></li><li><b>FLASH_ERROR_PGP</b>: <i>FLASH Programming Parallelism error flag</i></li><li><b>FLASH_ERROR_PGA</b>: <i>FLASH Programming Alignment error flag</i></li><li><b>FLASH_ERROR_WRP</b>: <i>FLASH Write protected error flag</i></li><li><b>FLASH_ERROR_OPERATION</b>: <i>FLASH operation Error flag</i></li></ul></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 17.3 FLASH Firmware driver defines

### 17.3.1 FLASH

FLASH

*FLASH\_Exported\_Constants*

- #define: **ACR\_BYTE0\_ADDRESS** ((*uint32\_t*)0x40023C00)
- #define: **OPTCR\_BYTE0\_ADDRESS** ((*uint32\_t*)0x40023C14)
- #define: **OPTCR\_BYTE1\_ADDRESS** ((*uint32\_t*)0x40023C15)
- #define: **OPTCR\_BYTE2\_ADDRESS** ((*uint32\_t*)0x40023C16)
- #define: **OPTCR\_BYTE3\_ADDRESS** ((*uint32\_t*)0x40023C17)
- #define: **OPTCR1\_BYTE2\_ADDRESS** ((*uint32\_t*)0x40023C1A)

*FLASH\_Flag\_definition*

- #define: **FLASH\_FLAG\_EOP** **FLASH\_SR\_EOP**  
*FLASH End of Operation flag*
- #define: **FLASH\_FLAG\_OPERR** **FLASH\_SR\_SOP**  
*FLASH operation Error flag*
- #define: **FLASH\_FLAG\_WRPERR** **FLASH\_SR\_WRPERR**  
*FLASH Write protected error flag*
- #define: **FLASH\_FLAG\_PGAERR** **FLASH\_SR\_PGAERR**  
*FLASH Programming Alignment error flag*
- #define: **FLASH\_FLAG\_PGPERR** **FLASH\_SR\_PGPERR**

*FLASH Programming Parallelism error flag*

- #define: **FLASH\_FLAG\_PGSERR** **FLASH\_SR\_PGSERR**  
*FLASH Programming Sequence error flag*

- #define: **FLASH\_FLAG\_RDERR** ((*uint32\_t*)0x00000100)  
*Read Protection error flag (PCROP)*

- #define: **FLASH\_FLAG\_BSY** **FLASH\_SR\_BSY**  
*FLASH Busy flag*

***FLASH Interrupt definition***

- #define: **FLASH\_IT\_EOP** **FLASH\_CR\_EOPIE**  
*End of FLASH Operation Interrupt source*

- #define: **FLASH\_IT\_ERR** ((*uint32\_t*)0x02000000)  
*Error Interrupt source*

***FLASH Keys***

- #define: **RDP\_KEY** ((*uint16\_t*)0x00A5)
- #define: **FLASH\_KEY1** ((*uint32\_t*)0x45670123)
- #define: **FLASH\_KEY2** ((*uint32\_t*)0xCDEF89AB)
- #define: **FLASH\_OPT\_KEY1** ((*uint32\_t*)0x08192A3B)
- #define: **FLASH\_OPT\_KEY2** ((*uint32\_t*)0x4C5D6E7F)

***FLASH Latency***

- #define: **FLASH\_LATENCY\_0** **FLASH\_ACR\_LATENCY\_0WS**

*FLASH Zero Latency cycle*

- #define: **FLASH\_LATENCY\_1 FLASH\_ACR\_LATENCY\_1WS**  
*FLASH One Latency cycle*
- #define: **FLASH\_LATENCY\_2 FLASH\_ACR\_LATENCY\_2WS**  
*FLASH Two Latency cycles*
- #define: **FLASH\_LATENCY\_3 FLASH\_ACR\_LATENCY\_3WS**  
*FLASH Three Latency cycles*
- #define: **FLASH\_LATENCY\_4 FLASH\_ACR\_LATENCY\_4WS**  
*FLASH Four Latency cycles*
- #define: **FLASH\_LATENCY\_5 FLASH\_ACR\_LATENCY\_5WS**  
*FLASH Five Latency cycles*
- #define: **FLASH\_LATENCY\_6 FLASH\_ACR\_LATENCY\_6WS**  
*FLASH Six Latency cycles*
- #define: **FLASH\_LATENCY\_7 FLASH\_ACR\_LATENCY\_7WS**  
*FLASH Seven Latency cycles*
- #define: **FLASH\_LATENCY\_8 FLASH\_ACR\_LATENCY\_8WS**  
*FLASH Eight Latency cycles*
- #define: **FLASH\_LATENCY\_9 FLASH\_ACR\_LATENCY\_9WS**  
*FLASH Nine Latency cycles*
- #define: **FLASH\_LATENCY\_10 FLASH\_ACR\_LATENCY\_10WS**  
*FLASH Ten Latency cycles*
- #define: **FLASH\_LATENCY\_11 FLASH\_ACR\_LATENCY\_11WS**  
*FLASH Eleven Latency cycles*
- #define: **FLASH\_LATENCY\_12 FLASH\_ACR\_LATENCY\_12WS**

*FLASH Twelve Latency cycles*

- #define: **FLASH\_LATENCY\_13** **FLASH\_ACR\_LATENCY\_13WS**  
*FLASH Thirteen Latency cycles*

- #define: **FLASH\_LATENCY\_14** **FLASH\_ACR\_LATENCY\_14WS**  
*FLASH Fourteen Latency cycles*

- #define: **FLASH\_LATENCY\_15** **FLASH\_ACR\_LATENCY\_15WS**  
*FLASH Fifteen Latency cycles*

***FLASH\_Program\_Parallelism***

- #define: **FLASH\_PSIZE\_BYTE** ((*uint32\_t*)0x00000000)
- #define: **FLASH\_PSIZE\_HALF\_WORD** ((*uint32\_t*)0x00000100)
- #define: **FLASH\_PSIZE\_WORD** ((*uint32\_t*)0x00000200)
- #define: **FLASH\_PSIZE\_DOUBLE\_WORD** ((*uint32\_t*)0x00000300)
- #define: **CR\_PSIZE\_MASK** ((*uint32\_t*)0xFFFFFCFF)

***FLASH\_Type\_Program***

- #define: **TYPEPROGRAM\_BYTE** ((*uint32\_t*)0x00)  
*Program byte (8-bit) at a specified address*
- #define: **TYPEPROGRAM\_HALFWORD** ((*uint32\_t*)0x01)  
*Program a half-word (16-bit) at a specified address*
- #define: **TYPEPROGRAM\_WORD** ((*uint32\_t*)0x02)  
*Program a word (32-bit) at a specified address*

- #define: **TYPEPROGRAM\_DOUBLEWORD ((uint32\_t)0x03)**  
*Program a double word (64-bit) at a specified address*

## 18 HAL FLASH Extension Driver

### 18.1 FLASHEx Firmware driver registers structures

#### 18.1.1 FLASH\_EraseInitTypeDef

*FLASH\_EraseInitTypeDef* is defined in the `stm32f4xx_hal_flash_ex.h`

##### Data Fields

- *uint32\_t TypeErase*
- *uint32\_t Banks*
- *uint32\_t Sector*
- *uint32\_t NbSectors*
- *uint32\_t VoltageRange*

##### Field Documentation

- *uint32\_t FLASH\_EraseInitTypeDef::TypeErase*
  - Mass erase or sector Erase. This parameter can be a value of [\*FLASHEx\\_Type\\_Erase\*](#)
- *uint32\_t FLASH\_EraseInitTypeDef::Banks*
  - Select banks to erase when Mass erase is enabled This parameter must be a value of [\*FLASHEx\\_Banks\*](#)
- *uint32\_t FLASH\_EraseInitTypeDef::Sector*
  - Initial FLASH sector to erase when Mass erase is disabled This parameter must be a value of [\*FLASHEx\\_Sectors\*](#)
- *uint32\_t FLASH\_EraseInitTypeDef::NbSectors*
  - Number of sectors to be erased. This parameter must be a value between 1 and (max number of sectors - value of Initial sector)
- *uint32\_t FLASH\_EraseInitTypeDef::VoltageRange*
  - The device voltage range which defines the erase parallelism This parameter must be a value of [\*FLASHEx\\_Voltage\\_Range\*](#)

#### 18.1.2 FLASH\_OBProgramInitTypeDef

*FLASH\_OBProgramInitTypeDef* is defined in the `stm32f4xx_hal_flash_ex.h`

##### Data Fields

- *uint32\_t OptionType*
- *uint32\_t WRPState*
- *uint32\_t WRPSector*
- *uint32\_t Banks*
- *uint32\_t RDPLevel*
- *uint32\_t BORLevel*
- *uint8\_t USERConfig*

## Field Documentation

- ***uint32\_t FLASH\_OBProgramInitTypeDef::OptionType***
  - Option byte to be configured. This parameter can be a value of **FLASHEx\_Option\_Type**
- ***uint32\_t FLASH\_OBProgramInitTypeDef::WRPState***
  - Write protection activation or deactivation. This parameter can be a value of **FLASHEx\_WRP\_State**
- ***uint32\_t FLASH\_OBProgramInitTypeDef::WRPSector***
  - WRPSector: specifies the sector(s) to be write protected. The value of this parameter depend on device used within the same series
- ***uint32\_t FLASH\_OBProgramInitTypeDef::Banks***
  - Select banks for WRP activation/deactivation of all sectors. This parameter must be a value of **FLASHEx\_Banks**
- ***uint32\_t FLASH\_OBProgramInitTypeDef::RDPLevel***
  - Set the read protection level. This parameter can be a value of **FLASHEx\_Option\_Bytes\_Read\_Protection**
- ***uint32\_t FLASH\_OBProgramInitTypeDef::BORLevel***
  - Set the BOR Level. This parameter can be a value of **FLASHEx\_BOR\_Reset\_Level**
- ***uint8\_t FLASH\_OBProgramInitTypeDef::USERConfig***
  - Program the FLASH User Option Byte: IWDG\_SW / RST\_STOP / RST\_STDBY. This parameter can be a combination of FLASH Option Bytes IWatchdog, FLASH Option Bytes nRST\_STOP and FLASH Option Bytes nRST\_STDBY

### 18.1.3 **FLASH\_AdvOBProgramInitTypeDef**

**FLASH\_AdvOBProgramInitTypeDef** is defined in the `stm32f4xx_hal_flash_ex.h`

#### Data Fields

- ***uint32\_t OptionType***
- ***uint32\_t PCROPState***
- ***uint16\_t Sectors***
- ***uint32\_t Banks***
- ***uint16\_t SectorsBank1***
- ***uint16\_t SectorsBank2***
- ***uint8\_t BootConfig***

## Field Documentation

- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::OptionType***
  - Option byte to be configured for extension . This parameter can be a value of **FLASHEx\_Advanced\_Option\_Type**
- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::PCROPState***
  - PCROP activation or deactivation. This parameter can be a value of **FLASHEx\_PCROP\_State**
- ***uint16\_t FLASH\_AdvOBProgramInitTypeDef::Sectors***

- specifies the sector(s) set for PCROP This parameter can be a value of ***FLASHEx\_Option\_Bytes\_PC\_ReadWrite\_Protection***
- ***uint32\_t FLASH\_AdvOBProgramInitTypeDef::Banks***
  - Select banks for PCROP activation/deactivation of all sectors This parameter must be a value of ***FLASHEx\_Banks***
- ***uint16\_t FLASH\_AdvOBProgramInitTypeDef::SectorsBank1***
  - Specifies the sector(s) set for PCROP for Bank1 This parameter can be a value of ***FLASHEx\_Option\_Bytes\_PC\_ReadWrite\_Protection***
- ***uint16\_t FLASH\_AdvOBProgramInitTypeDef::SectorsBank2***
  - Specifies the sector(s) set for PCROP for Bank2 This parameter can be a value of ***FLASHEx\_Option\_Bytes\_PC\_ReadWrite\_Protection***
- ***uint8\_t FLASH\_AdvOBProgramInitTypeDef::BootConfig***
  - Specifies Option bytes for boot config This parameter can be a value of ***FLASHEx\_Dual\_Boot***

## 18.2 FLASHEx Firmware driver API description

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

### 18.2.1 Flash Extension features

Comparing to other previous devices, the FLASH interface for STM32F427xx/437xx and STM32F429xx/439xx devices contains the following additional features

- Capacity up to 2 Mbyte with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank memory organization
- PCROP protection for all banks

### 18.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32F427xx/437xx and STM32F429xx/439xx devices. It includes

1. FLASH Memory Erase functions:
  - Lock and Unlock the FLASH interface using HAL\_FLASH\_Unlock() and HAL\_FLASH\_Lock() functions
  - Erase function: Erase sector, erase all sectors
  - There are two modes of erase :
    - Polling Mode using HAL\_FLASHEx\_Erase()
    - Interrupt Mode using HAL\_FLASHEx\_Erase\_IT()
2. Option Bytes Programming functions: Use HAL\_FLASHEx\_OBProgram() to :
  - Set/Reset the write protection
  - Set the Read protection Level
  - Set the BOR level
  - Program the user Option Bytes
3. Advanced Option Bytes Programming functions: Use HAL\_FLASHEx\_AdvOBProgram() to :
  - Extended space (bank 2) erase function
  - Full FLASH space (2 Mo) erase (bank 1 and bank 2)
  - Dual Boot activation

- Write protection configuration for bank 2
- PCROP protection configuration and control for both banks

### 18.2.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extension FLASH programming operations Operations.

- [`HAL\_FLASHEx\_Erase\(\)`](#)
- [`HAL\_FLASHEx\_Erase\_IT\(\)`](#)
- [`HAL\_FLASHEx\_OBProgram\(\)`](#)
- [`HAL\_FLASHEx\_OBGetConfig\(\)`](#)
- [`HAL\_FLASHEx\_AdvOBProgram\(\)`](#)
- [`HAL\_FLASHEx\_AdvOBGetConfig\(\)`](#)
- [`HAL\_FLASHEx\_OB\_SelectPCROP\(\)`](#)
- [`HAL\_FLASHEx\_OB\_DeSelectPCROP\(\)`](#)
- [`HAL\_FLASHEx\_OB\_GetBank2WRP\(\)`](#)

### 18.2.4 Extended IO operation functions

#### 18.2.4.1 HAL\_FLASHEx\_Erase

Function Name	<code>HAL_StatusTypeDef HAL_FLASHEx_Erase (</code> <a href="#"><code>FLASH_EraselInitTypeDef * pEraselInit</code></a> <code>, uint32_t * SectorError)</code>
Function Description	Perform a mass erase or erase the specified FLASH memory sectors.
Parameters	<ul style="list-style-type: none"> <li>• <b>pEraselInit</b> : pointer to an <code>FLASH_EraselInitTypeDef</code> structure that contains the configuration information for the erasing.</li> <li>• <b>SectorError</b> : pointer to variable that contains the configuration information on faulty sector in case of error (0xFFFFFFFF means that all the sectors have been correctly erased)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL Status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 18.2.4.2 HAL\_FLASHEx\_Erase\_IT

Function Name	<code>HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (</code> <a href="#"><code>FLASH_EraselInitTypeDef * pEraselInit</code></a> <code>)</code>
Function Description	Perform a mass erase or erase the specified FLASH memory

---

	sectors with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>pEraseInit</b> : pointer to an <code>FLASH_EraseInitTypeDef</code> structure that contains the configuration information for the erasing.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL Status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 18.2.4.3 HAL\_FLASHEx\_OBProgram

Function Name	<code>HAL_StatusTypeDef HAL_FLASHEx_OBProgram (</code> <code>FLASH_OBProgramInitTypeDef * pOBInit)</code>
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> <li>• <b>pOBInit</b> : pointer to an <code>FLASH_OBInitStruct</code> structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL Status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 18.2.4.4 HAL\_FLASHEx\_OBGetConfig

Function Name	<code>void HAL_FLASHEx_OBGetConfig (</code> <code>FLASH_OBProgramInitTypeDef * pOBInit)</code>
Function Description	Get the Option byte configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>pOBInit</b> : pointer to an <code>FLASH_OBInitStruct</code> structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 18.2.4.5 HAL\_FLASHEx\_AdvOBProgram

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_AdvOBProgram (</b> <b>FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)</b>
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> <li>• <b>pAdvOBInit</b> : pointer to an <b>FLASH_AdvOBProgramInitTypeDef</b> structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL Status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 18.2.4.6 HAL\_FLASHEx\_AdvOBGetConfig

Function Name	<b>void HAL_FLASHEx_AdvOBGetConfig (</b> <b>FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)</b>
Function Description	Get the OBEX byte configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>pAdvOBInit</b> : pointer to an <b>FLASH_AdvOBProgramInitTypeDef</b> structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 18.2.4.7 HAL\_FLASHEx\_OB\_SelectPCROP

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OB_SelectPCROP ( void )</b>
Function Description	Select the Protection Mode.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL Status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• After PCROP activated Option Byte modification NOT POSSIBLE! excepted Global Read Out Protection modification (from level1 to level0)</li> <li>• Once SPRMOD bit is active unprotection of a protected sector is not possible</li> <li>• Read a protected sector will set RDERR Flag and write a protected</li> </ul>

- sector will set WRPERR Flag
- This function can be used only for STM32F427xx/STM32F429xx/STM32F437xx/STM32F439xx/STM32F401xx devices.

#### 18.2.4.8 HAL\_FLASHEx\_OB\_DeSelectPCROP

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OB_DeSelectPCROP ( void )</b>
Function Description	Deselect the Protection Mode.
Parameters	<ul style="list-style-type: none"> <li>None.</li> </ul>
Return values	<b>HAL Status</b>
Notes	<ul style="list-style-type: none"> <li>After PCROP activated Option Byte modification NOT POSSIBLE! excepted Global Read Out Protection modification (from level1 to level0)</li> <li>Once SPRMOD bit is active unprotection of a protected sector is not possible</li> <li>Read a protected sector will set RDERR Flag and write a protected sector will set WRPERR Flag</li> <li>This function can be used only for STM32F427xx/STM32F429xx/STM32F437xx/STM32F439xx/STM32F401xx devices.</li> </ul>

#### 18.2.4.9 HAL\_FLASHEx\_OB\_GetBank2WRP

Function Name	<b>uint16_t HAL_FLASHEx_OB_GetBank2WRP ( void )</b>
Function Description	Returns the FLASH Write Protection Option Bytes value for Bank 2.
Parameters	<ul style="list-style-type: none"> <li>None.</li> </ul>
Return values	<b>The FLASH Write Protection Option Bytes value</b>
Notes	<ul style="list-style-type: none"> <li>This function can be used only for STM32F427X and STM32F429X devices.</li> </ul>

## 18.3 FLASHEx Firmware driver defines

### 18.3.1 FLASHEx

FLASHEx

***FLASHEx\_Advanced\_Option\_Type***

- #define: ***OBEX\_PCROP*** ((*uint32\_t*)0x01)

*PCROP option byte configuration*

- #define: ***OBEX\_BOOTCONFIG*** ((*uint32\_t*)0x02)

*BOOTConfig option byte configuration*

***FLASHEx\_Banks***

- #define: ***FLASH\_BANK\_1*** ((*uint32\_t*)1)

*Bank 1*

- #define: ***FLASH\_BANK\_2*** ((*uint32\_t*)2)

*Bank 2*

- #define: ***FLASH\_BANK\_BOTH*** ((*uint32\_t*)*FLASH\_BANK\_1* | *FLASH\_BANK\_2*)

*Bank1 and Bank2*

***FLASHEx\_BOR\_Reset\_Level***

- #define: ***OB\_BOR\_LEVEL3*** ((*uint8\_t*)0x00)

*Supply voltage ranges from 2.70 to 3.60 V*

- #define: ***OB\_BOR\_LEVEL2*** ((*uint8\_t*)0x04)

*Supply voltage ranges from 2.40 to 2.70 V*

- #define: ***OB\_BOR\_LEVEL1*** ((*uint8\_t*)0x08)

*Supply voltage ranges from 2.10 to 2.40 V*

- #define: ***OB\_BOR\_OFF*** ((*uint8\_t*)0x0C)

*Supply voltage ranges from 1.62 to 2.10 V*

***FLASHEx\_Dual\_Boot***

- #define: **OB\_DUAL\_BOOT\_ENABLE** ((*uint8\_t*)0x10)

*Dual Bank Boot Enable*

- #define: **OB\_DUAL\_BOOT\_DISABLE** ((*uint8\_t*)0x00)

*Dual Bank Boot Disable, always boot on User Flash*

#### ***FLASHEx\_MassErase\_bit***

- #define: **FLASH\_MER\_BIT** (*FLASH\_CR\_MER1* | *FLASH\_CR\_MER2*)

*2 MER bits here to clear*

#### ***FLASHEx\_Option\_Bytes\_IWatchdog***

- #define: **OB\_IWDG\_SW** ((*uint8\_t*)0x20)

*Software IWDG selected*

- #define: **OB\_IWDG\_HW** ((*uint8\_t*)0x00)

*Hardware IWDG selected*

#### ***FLASHEx\_Option\_Bytes\_nRST\_STDBY***

- #define: **OB\_STDBY\_NO\_RST** ((*uint8\_t*)0x80)

*No reset generated when entering in STANDBY*

- #define: **OB\_STDBY\_RST** ((*uint8\_t*)0x00)

*Reset generated when entering in STANDBY*

#### ***FLASHEx\_Option\_Bytes\_nRST\_STOP***

- #define: **OB\_STOP\_NO\_RST** ((*uint8\_t*)0x40)

*No reset generated when entering in STOP*

- #define: **OB\_STOP\_RST** ((*uint8\_t*)0x00)

*Reset generated when entering in STOP*

#### ***FLASHEx\_Option\_Bytes\_PC\_ReadWrite\_Protection***

- #define: **OB\_PCROP\_SECTOR\_0** ((*uint32\_t*)0x00000001)

*PC Read/Write protection of Sector0*

- #define: **OB\_PCROP\_SECTOR\_1** ((*uint32\_t*)0x00000002)

*PC Read/Write protection of Sector1*

- #define: ***OB\_PCROP\_SECTOR\_2 ((uint32\_t)0x00000004)***

*PC Read/Write protection of Sector2*

- #define: ***OB\_PCROP\_SECTOR\_3 ((uint32\_t)0x00000008)***

*PC Read/Write protection of Sector3*

- #define: ***OB\_PCROP\_SECTOR\_4 ((uint32\_t)0x00000010)***

*PC Read/Write protection of Sector4*

- #define: ***OB\_PCROP\_SECTOR\_5 ((uint32\_t)0x00000020)***

*PC Read/Write protection of Sector5*

- #define: ***OB\_PCROP\_SECTOR\_6 ((uint32\_t)0x00000040)***

*PC Read/Write protection of Sector6*

- #define: ***OB\_PCROP\_SECTOR\_7 ((uint32\_t)0x00000080)***

*PC Read/Write protection of Sector7*

- #define: ***OB\_PCROP\_SECTOR\_8 ((uint32\_t)0x00000100)***

*PC Read/Write protection of Sector8*

- #define: ***OB\_PCROP\_SECTOR\_9 ((uint32\_t)0x00000200)***

*PC Read/Write protection of Sector9*

- #define: ***OB\_PCROP\_SECTOR\_10 ((uint32\_t)0x00000400)***

*PC Read/Write protection of Sector10*

- #define: ***OB\_PCROP\_SECTOR\_11 ((uint32\_t)0x00000800)***

*PC Read/Write protection of Sector11*

- #define: ***OB\_PCROP\_SECTOR\_12 ((uint32\_t)0x00000001)***

*PC Read/Write protection of Sector12*

- #define: ***OB\_PCROP\_SECTOR\_13 ((uint32\_t)0x00000002)***

*PC Read/Write protection of Sector13*

- #define: ***OB\_PCROP\_SECTOR\_14 ((uint32\_t)0x00000004)***

*PC Read/Write protection of Sector14*

- #define: ***OB\_PCROP\_SECTOR\_15 ((uint32\_t)0x00000008)***

*PC Read/Write protection of Sector15*

- #define: ***OB\_PCROP\_SECTOR\_16 ((uint32\_t)0x00000010)***

*PC Read/Write protection of Sector16*

- #define: ***OB\_PCROP\_SECTOR\_17 ((uint32\_t)0x00000020)***

*PC Read/Write protection of Sector17*

- #define: ***OB\_PCROP\_SECTOR\_18 ((uint32\_t)0x00000040)***

*PC Read/Write protection of Sector18*

- #define: ***OB\_PCROP\_SECTOR\_19 ((uint32\_t)0x00000080)***

*PC Read/Write protection of Sector19*

- #define: ***OB\_PCROP\_SECTOR\_20 ((uint32\_t)0x00000100)***

*PC Read/Write protection of Sector20*

- #define: ***OB\_PCROP\_SECTOR\_21 ((uint32\_t)0x00000200)***

*PC Read/Write protection of Sector21*

- #define: ***OB\_PCROP\_SECTOR\_22 ((uint32\_t)0x00000400)***

*PC Read/Write protection of Sector22*

- #define: ***OB\_PCROP\_SECTOR\_23 ((uint32\_t)0x00000800)***

*PC Read/Write protection of Sector23*

- #define: ***OB\_PCROP\_SECTOR\_All ((uint32\_t)0x00000FFF)***

*PC Read/Write protection of all Sectors*

#### ***FLASHEx\_Option\_Bytes\_Read\_Protection***

- #define: ***OB\_RDP\_LEVEL\_0*** ((*uint8\_t*)0xAA)

- #define: ***OB\_RDP\_LEVEL\_1*** ((*uint8\_t*)0x55)

#### ***FLASHEx\_Option\_Bytes\_Write\_Protection***

- #define: ***OB\_WRP\_SECTOR\_0*** ((*uint32\_t*)0x00000001)

*Write protection of Sector0*

- #define: ***OB\_WRP\_SECTOR\_1*** ((*uint32\_t*)0x00000002)

*Write protection of Sector1*

- #define: ***OB\_WRP\_SECTOR\_2*** ((*uint32\_t*)0x00000004)

*Write protection of Sector2*

- #define: ***OB\_WRP\_SECTOR\_3*** ((*uint32\_t*)0x00000008)

*Write protection of Sector3*

- #define: ***OB\_WRP\_SECTOR\_4*** ((*uint32\_t*)0x00000010)

*Write protection of Sector4*

- #define: ***OB\_WRP\_SECTOR\_5*** ((*uint32\_t*)0x00000020)

*Write protection of Sector5*

- #define: ***OB\_WRP\_SECTOR\_6*** ((*uint32\_t*)0x00000040)

*Write protection of Sector6*

- #define: ***OB\_WRP\_SECTOR\_7*** ((*uint32\_t*)0x00000080)

*Write protection of Sector7*

- #define: ***OB\_WRP\_SECTOR\_8*** ((*uint32\_t*)0x00000100)

*Write protection of Sector8*

- #define: ***OB\_WRP\_SECTOR\_9*** ((*uint32\_t*)0x00000200)

*Write protection of Sector9*

- #define: ***OB\_WRP\_SECTOR\_10*** ((*uint32\_t*)0x00000400)  
*Write protection of Sector10*
  
- #define: ***OB\_WRP\_SECTOR\_11*** ((*uint32\_t*)0x00000800)  
*Write protection of Sector11*
  
- #define: ***OB\_WRP\_SECTOR\_12*** ((*uint32\_t*)0x00000001 << 12)  
*Write protection of Sector12*
  
- #define: ***OB\_WRP\_SECTOR\_13*** ((*uint32\_t*)0x00000002 << 12)  
*Write protection of Sector13*
  
- #define: ***OB\_WRP\_SECTOR\_14*** ((*uint32\_t*)0x00000004 << 12)  
*Write protection of Sector14*
  
- #define: ***OB\_WRP\_SECTOR\_15*** ((*uint32\_t*)0x00000008 << 12)  
*Write protection of Sector15*
  
- #define: ***OB\_WRP\_SECTOR\_16*** ((*uint32\_t*)0x00000010 << 12)  
*Write protection of Sector16*
  
- #define: ***OB\_WRP\_SECTOR\_17*** ((*uint32\_t*)0x00000020 << 12)  
*Write protection of Sector17*
  
- #define: ***OB\_WRP\_SECTOR\_18*** ((*uint32\_t*)0x00000040 << 12)  
*Write protection of Sector18*
  
- #define: ***OB\_WRP\_SECTOR\_19*** ((*uint32\_t*)0x00000080 << 12)  
*Write protection of Sector19*
  
- #define: ***OB\_WRP\_SECTOR\_20*** ((*uint32\_t*)0x00000100 << 12)  
*Write protection of Sector20*
  
- #define: ***OB\_WRP\_SECTOR\_21*** ((*uint32\_t*)0x00000200 << 12)  
*Write protection of Sector21*

- #define: **OB\_WRP\_SECTOR\_22** ((*uint32\_t*)0x00000400 << 12)

*Write protection of Sector22*

- #define: **OB\_WRP\_SECTOR\_23** ((*uint32\_t*)0x00000800 << 12)

*Write protection of Sector23*

- #define: **OB\_WRP\_SECTOR\_All** ((*uint32\_t*)0x00000FFF << 12)

*Write protection of all Sectors*

#### **FLASHEx\_Option\_Type**

- #define: **OPTIONBYTE\_WRP** ((*uint32\_t*)0x01)

*WRP option byte configuration*

- #define: **OPTIONBYTE\_RDP** ((*uint32\_t*)0x02)

*RDP option byte configuration*

- #define: **OPTIONBYTE\_USER** ((*uint32\_t*)0x04)

*USER option byte configuration*

- #define: **OPTIONBYTE\_BOR** ((*uint32\_t*)0x08)

*BOR option byte configuration*

#### **FLASHEx\_PCROP\_State**

- #define: **PCROPSTATE\_DISABLE** ((*uint32\_t*)0x00)

*Disable PCROP*

- #define: **PCROPSTATE\_ENABLE** ((*uint32\_t*)0x01)

*Enable PCROP*

#### **FLASHEx\_Sectors**

- #define: **FLASH\_SECTOR\_0** ((*uint32\_t*)0)

*Sector Number 0*

- #define: **FLASH\_SECTOR\_1** ((*uint32\_t*)1)

*Sector Number 1*

- #define: **FLASH\_SECTOR\_2** ((*uint32\_t*)2)  
*Sector Number 2*
  
- #define: **FLASH\_SECTOR\_3** ((*uint32\_t*)3)  
*Sector Number 3*
  
- #define: **FLASH\_SECTOR\_4** ((*uint32\_t*)4)  
*Sector Number 4*
  
- #define: **FLASH\_SECTOR\_5** ((*uint32\_t*)5)  
*Sector Number 5*
  
- #define: **FLASH\_SECTOR\_6** ((*uint32\_t*)6)  
*Sector Number 6*
  
- #define: **FLASH\_SECTOR\_7** ((*uint32\_t*)7)  
*Sector Number 7*
  
- #define: **FLASH\_SECTOR\_8** ((*uint32\_t*)8)  
*Sector Number 8*
  
- #define: **FLASH\_SECTOR\_9** ((*uint32\_t*)9)  
*Sector Number 9*
  
- #define: **FLASH\_SECTOR\_10** ((*uint32\_t*)10)  
*Sector Number 10*
  
- #define: **FLASH\_SECTOR\_11** ((*uint32\_t*)11)  
*Sector Number 11*
  
- #define: **FLASH\_SECTOR\_12** ((*uint32\_t*)12)  
*Sector Number 12*
  
- #define: **FLASH\_SECTOR\_13** ((*uint32\_t*)13)  
*Sector Number 13*

- #define: **FLASH\_SECTOR\_14** ((*uint32\_t*)14)  
*Sector Number 14*
- #define: **FLASH\_SECTOR\_15** ((*uint32\_t*)15)  
*Sector Number 15*
- #define: **FLASH\_SECTOR\_16** ((*uint32\_t*)16)  
*Sector Number 16*
- #define: **FLASH\_SECTOR\_17** ((*uint32\_t*)17)  
*Sector Number 17*
- #define: **FLASH\_SECTOR\_18** ((*uint32\_t*)18)  
*Sector Number 18*
- #define: **FLASH\_SECTOR\_19** ((*uint32\_t*)19)  
*Sector Number 19*
- #define: **FLASH\_SECTOR\_20** ((*uint32\_t*)20)  
*Sector Number 20*
- #define: **FLASH\_SECTOR\_21** ((*uint32\_t*)21)  
*Sector Number 21*
- #define: **FLASH\_SECTOR\_22** ((*uint32\_t*)22)  
*Sector Number 22*
- #define: **FLASH\_SECTOR\_23** ((*uint32\_t*)23)  
*Sector Number 23*
- #define: **FLASH\_SECTOR\_TOTAL** 24

***FLASHEx\_Selection\_Protection\_Mode***

- #define: **OB\_PCROP\_DESELECTED** ((*uint8\_t*)0x00)

*Disabled PcROP, nWPRi bits used for Write Protection on sector i*

- #define: **OB\_PCRROP\_SELECTED ((uint8\_t)0x80)**

*Enable PcROP, nWPRi bits used for PCRoP Protection on sector i*

#### ***FLASHEx\_Type\_Erase***

- #define: **TYPEERASE\_SECTORS ((uint32\_t)0x00)**

*Sectors erase only*

- #define: **TYPEERASE\_MASSERASE ((uint32\_t)0x01)**

*Flash Mass erase activation*

#### ***FLASHEx\_Voltage\_Range***

- #define: **VOLTAGE\_RANGE\_1 ((uint32\_t)0x00)**

*Device operating range: 1.8V to 2.1V*

- #define: **VOLTAGE\_RANGE\_2 ((uint32\_t)0x01)**

*Device operating range: 2.1V to 2.7V*

- #define: **VOLTAGE\_RANGE\_3 ((uint32\_t)0x02)**

*Device operating range: 2.7V to 3.6V*

- #define: **VOLTAGE\_RANGE\_4 ((uint32\_t)0x03)**

*Device operating range: 2.7V to 3.6V + External Vpp*

#### ***FLASHEx\_WRP\_State***

- #define: **WRPSTATE\_DISABLE ((uint32\_t)0x00)**

*Disable the write protection of the desired bank 1 sectors*

- #define: **WRPSTATE\_ENABLE ((uint32\_t)0x01)**

*Enable the write protection of the desired bank 1 sectors*

## 19 HAL GPIO Generic Driver

### 19.1 GPIO Firmware driver registers structures

#### 19.1.1 GPIO\_InitTypeDef

*GPIO\_InitTypeDef* is defined in the `stm32f4xx_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 [\*GPIO\\_Alternat\\_function\\_selection\*](#)

#### 19.1.2 GPIO\_TypeDef

*GPIO\_TypeDef* is defined in the `stm32f439xx.h`

##### Data Fields

- *\_\_IO uint32\_t MODER*
- *\_\_IO uint32\_t OTYPER*
- *\_\_IO uint32\_t OSPEEDR*
- *\_\_IO uint32\_t PUPDR*
- *\_\_IO uint32\_t IDR*
- *\_\_IO uint32\_t ODR*
- *\_\_IO uint16\_t BSRRRL*
- *\_\_IO uint16\_t BSRRRH*

- `__IO uint32_t LCKR`
- `__IO uint32_t AFR`

#### Field Documentation

- `__IO uint32_t GPIO_TypeDef::MODER`
  - GPIO port mode register, Address offset: 0x00
- `__IO uint32_t GPIO_TypeDef::OTYPER`
  - GPIO port output type register, Address offset: 0x04
- `__IO uint32_t GPIO_TypeDef::OSPEEDR`
  - GPIO port output speed register, Address offset: 0x08
- `__IO uint32_t GPIO_TypeDef::PUPDR`
  - GPIO port pull-up/pull-down register, Address offset: 0x0C
- `__IO uint32_t GPIO_TypeDef::IDR`
  - GPIO port input data register, Address offset: 0x10
- `__IO uint32_t GPIO_TypeDef::ODR`
  - GPIO port output data register, Address offset: 0x14
- `__IO uint16_t GPIO_TypeDef::BSRRL`
  - GPIO port bit set/reset low register, Address offset: 0x18
- `__IO uint16_t GPIO_TypeDef::BSRRH`
  - GPIO port bit set/reset high register, Address offset: 0x1A
- `__IO uint32_t GPIO_TypeDef::LCKR`
  - GPIO port configuration lock register, Address offset: 0x1C
- `__IO uint32_t GPIO_TypeDef::AFR[2]`
  - GPIO alternate function registers, Address offset: 0x20-0x24

## 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 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.
  - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from `GPIO_InitTypeDef` structure.
  - Analog mode is required when a pin is to be used as ADC channel or DAC output.
  - In case of external interrupt/event selection the "Mode" member from `GPIO_InitTypeDef` structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using `HAL_NVIC_SetPriority()` and enable it using `HAL_NVIC_EnableIRQ()`.
4. To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()`.
5. To set/reset the level of a pin configured in output mode use `HAL_GPIO_WritePin()`/`HAL_GPIO_TogglePin()`.
6. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
7. 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.
8. The HSE oscillator pins OSC\_IN/OSC\_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

### 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\\_EXTI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_GPIO\\_EXTI\\_Callback\(\)\*](#)

### 19.2.5 Initialization and de-initialization functions

#### 19.2.5.1 HAL\_GPIO\_Init

Function Name	<b>void HAL_GPIO_Init ( <i>GPIO_TypeDef</i> * GPIOx, <i>GPIO_InitTypeDef</i> * GPIO_InitStruct)</b>
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b> : where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.</li> <li>• <b>GPIO_InitStruct</b> : pointer to a <i>GPIO_InitTypeDef</i> structure that contains the configuration information for the specified GPIO peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 19.2.5.2 HAL\_GPIO\_DeInit

Function Name	<b>void HAL_GPIO_DeInit ( <i>GPIO_TypeDef</i> * GPIOx, uint32_t GPIO_Pin)</b>
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b> : where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.</li> <li>• <b>GPIO_Pin</b> : specifies the port bit to be written. This parameter can be one of <i>GPIO_PIN_x</i> where x can be (0..15).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 19.2.6 IO operation functions

### 19.2.6.1 HAL\_GPIO\_ReadPin

Function Name	<b>GPIO_PinState HAL_GPIO_ReadPin ( <i>GPIO_TypeDef</i> * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Reads the specified input port pin.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx</b> : where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.</li> <li><b>GPIO_Pin</b> : specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>The input port pin value.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 19.2.6.2 HAL\_GPIO\_WritePin

Function Name	<b>void HAL_GPIO_WritePin ( <i>GPIO_TypeDef</i> * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)</b>
Function Description	Sets or clears the selected data port bit.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx</b> : where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.</li> <li><b>GPIO_Pin</b> : specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).</li> <li><b>PinState</b> : specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_BIT_RESET: to clear the port pin GPIO_BIT_SET: to set the port pin</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> </ul>

### 19.2.6.3 HAL\_GPIO\_TogglePin

Function Name	<b>void HAL_GPIO_TogglePin ( <i>GPIO_TypeDef</i> * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Toggles the specified GPIO pins.
Parameters	<ul style="list-style-type: none"><li>• <b>GPIOx</b> : Where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.</li><li>• <b>GPIO_Pin</b> : Specifies the pins to be toggled.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 19.2.6.4 HAL\_GPIO\_EXTI\_IRQHandler

Function Name	<b>void HAL_GPIO_EXTI_IRQHandler ( uint16_t GPIO_Pin)</b>
Function Description	This function handles EXTI interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>GPIO_Pin</b> : Specifies the pins connected EXTI line</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 19.2.6.5 HAL\_GPIO\_EXTI\_Callback

Function Name	<b>void HAL_GPIO_EXTI_Callback ( uint16_t GPIO_Pin)</b>
Function Description	EXTI line detection callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>GPIO_Pin</b> : Specifies the pins connected EXTI line</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 19.3 GPIO Firmware driver defines

### 19.3.1 GPIO

GPIO

*GPIO\_Alternat\_function\_selection*

- #define: ***GPIO\_AF0\_RTC\_50Hz*** ((*uint8\_t*)0x00)
- #define: ***GPIO\_AF0\_MCO*** ((*uint8\_t*)0x00)
- #define: ***GPIO\_AF0\_TAMPER*** ((*uint8\_t*)0x00)
- #define: ***GPIO\_AF0\_SWJ*** ((*uint8\_t*)0x00)
- #define: ***GPIO\_AF0\_TRACE*** ((*uint8\_t*)0x00)
- #define: ***GPIO\_AF1\_TIM1*** ((*uint8\_t*)0x01)
- #define: ***GPIO\_AF1\_TIM2*** ((*uint8\_t*)0x01)
- #define: ***GPIO\_AF2\_TIM3*** ((*uint8\_t*)0x02)
- #define: ***GPIO\_AF2\_TIM4*** ((*uint8\_t*)0x02)
- #define: ***GPIO\_AF2\_TIM5*** ((*uint8\_t*)0x02)
- #define: ***GPIO\_AF3\_TIM8*** ((*uint8\_t*)0x03)

- #define: ***GPIO\_AF3\_TIM9 ((uint8\_t)0x03)***
- #define: ***GPIO\_AF3\_TIM10 ((uint8\_t)0x03)***
- #define: ***GPIO\_AF3\_TIM11 ((uint8\_t)0x03)***
- #define: ***GPIO\_AF4\_I2C1 ((uint8\_t)0x04)***
- #define: ***GPIO\_AF4\_I2C2 ((uint8\_t)0x04)***
- #define: ***GPIO\_AF4\_I2C3 ((uint8\_t)0x04)***
- #define: ***GPIO\_AF5\_SPI1 ((uint8\_t)0x05)***
- #define: ***GPIO\_AF5\_SPI2 ((uint8\_t)0x05)***
- #define: ***GPIO\_AF5\_SPI4 ((uint8\_t)0x05)***
- #define: ***GPIO\_AF5\_SPI5 ((uint8\_t)0x05)***
- #define: ***GPIO\_AF5\_SPI6 ((uint8\_t)0x05)***
- #define: ***GPIO\_AF5\_I2S3ext ((uint8\_t)0x05)***

- #define: ***GPIO\_AF6\_SPI3 ((uint8\_t)0x06)***
- #define: ***GPIO\_AF6\_I2S2ext ((uint8\_t)0x06)***
- #define: ***GPIO\_AF6\_SAI1 ((uint8\_t)0x06)***
- #define: ***GPIO\_AF7\_USART1 ((uint8\_t)0x07)***
- #define: ***GPIO\_AF7\_USART2 ((uint8\_t)0x07)***
- #define: ***GPIO\_AF7\_USART3 ((uint8\_t)0x07)***
- #define: ***GPIO\_AF7\_I2S3ext ((uint8\_t)0x07)***
- #define: ***GPIO\_AF8\_UART4 ((uint8\_t)0x08)***
- #define: ***GPIO\_AF8\_UART5 ((uint8\_t)0x08)***
- #define: ***GPIO\_AF8\_USART6 ((uint8\_t)0x08)***
- #define: ***GPIO\_AF8\_UART7 ((uint8\_t)0x08)***
- #define: ***GPIO\_AF8\_UART8 ((uint8\_t)0x08)***

- #define: ***GPIO\_AF9\_CAN1*** ((*uint8\_t*)0x09)
- #define: ***GPIO\_AF9\_CAN2*** ((*uint8\_t*)0x09)
- #define: ***GPIO\_AF9\_TIM12*** ((*uint8\_t*)0x09)
- #define: ***GPIO\_AF9\_TIM13*** ((*uint8\_t*)0x09)
- #define: ***GPIO\_AF9\_LTDC*** ((*uint8\_t*)0x09)
- #define: ***GPIO\_AF10\_OTG\_FS*** ((*uint8\_t*)0xA)
- #define: ***GPIO\_AF10\_OTG\_HS*** ((*uint8\_t*)0xA)
- #define: ***GPIO\_AF11\_ETH*** ((*uint8\_t*)0xB)
- #define: ***GPIO\_AF12\_FMC*** ((*uint8\_t*)0xC)
- #define: ***GPIO\_AF12\_OTG\_HS\_FS*** ((*uint8\_t*)0xC)
- #define: ***GPIO\_AF12\_SDIO*** ((*uint8\_t*)0xC)

- #define: **GPIO\_AF13\_DCMI** ((*uint8\_t*)0x0D)
  - #define: **GPIO\_AF14\_LTDC** ((*uint8\_t*)0x0E)
  - #define: **GPIO\_AF15\_EVENTOUT** ((*uint8\_t*)0x0F)
- 
- GPIO\_mode\_define**
- #define: **GPIO\_MODE\_INPUT** ((*uint32\_t*)0x00000000)  
*Input Floating Mode*
  - #define: **GPIO\_MODE\_OUTPUT\_PP** ((*uint32\_t*)0x00000001)  
*Output Push Pull Mode*
  - #define: **GPIO\_MODE\_OUTPUT\_OD** ((*uint32\_t*)0x00000011)  
*Output Open Drain Mode*
  - #define: **GPIO\_MODE\_AF\_PP** ((*uint32\_t*)0x00000002)  
*Alternate Function Push Pull Mode*
  - #define: **GPIO\_MODE\_AF\_OD** ((*uint32\_t*)0x00000012)  
*Alternate Function Open Drain Mode*
  - #define: **GPIO\_MODE\_ANALOG** ((*uint32\_t*)0x00000003)  
*Analog Mode*
  - #define: **GPIO\_MODE\_IT\_RISING** ((*uint32\_t*)0x10110000)  
*External Interrupt Mode with Rising edge trigger detection*
  - #define: **GPIO\_MODE\_IT\_FALLING** ((*uint32\_t*)0x10210000)  
*External Interrupt Mode with Falling edge trigger detection*
  - #define: **GPIO\_MODE\_IT\_RISING\_FALLING** ((*uint32\_t*)0x10310000)

*External Interrupt Mode with Rising/Falling edge trigger detection*

- #define: **GPIO\_MODE\_EVT\_RISING** ((*uint32\_t*)0x10120000)

*External Event Mode with Rising edge trigger detection*

- #define: **GPIO\_MODE\_EVT\_FALLING** ((*uint32\_t*)0x10220000)

*External Event Mode with Falling edge trigger detection*

- #define: **GPIO\_MODE\_EVT\_RISING\_FALLING** ((*uint32\_t*)0x10320000)

*External Event Mode with Rising/Falling edge trigger detection*

#### **GPIO\_pins\_define**

- #define: **GPIO\_PIN\_0** ((*uint16\_t*)0x0001)

- #define: **GPIO\_PIN\_1** ((*uint16\_t*)0x0002)

- #define: **GPIO\_PIN\_2** ((*uint16\_t*)0x0004)

- #define: **GPIO\_PIN\_3** ((*uint16\_t*)0x0008)

- #define: **GPIO\_PIN\_4** ((*uint16\_t*)0x0010)

- #define: **GPIO\_PIN\_5** ((*uint16\_t*)0x0020)

- #define: **GPIO\_PIN\_6** ((*uint16\_t*)0x0040)

- #define: **GPIO\_PIN\_7** ((*uint16\_t*)0x0080)

- #define: **GPIO\_PIN\_8** ((*uint16\_t*)0x0100)
- #define: **GPIO\_PIN\_9** ((*uint16\_t*)0x0200)
- #define: **GPIO\_PIN\_10** ((*uint16\_t*)0x0400)
- #define: **GPIO\_PIN\_11** ((*uint16\_t*)0x0800)
- #define: **GPIO\_PIN\_12** ((*uint16\_t*)0x1000)
- #define: **GPIO\_PIN\_13** ((*uint16\_t*)0x2000)
- #define: **GPIO\_PIN\_14** ((*uint16\_t*)0x4000)
- #define: **GPIO\_PIN\_15** ((*uint16\_t*)0x8000)
- #define: **GPIO\_PIN\_All** ((*uint16\_t*)0xFFFF)

#### **GPIO\_pull\_define**

- #define: **GPIO\_NOPULL** ((*uint32\_t*)0x00000000)

*No Pull-up or Pull-down activation*

- #define: **GPIO\_PULLUP** ((*uint32\_t*)0x00000001)

*Pull-up activation*

- #define: **GPIO\_PULLDOWN** ((*uint32\_t*)0x00000002)

*Pull-down activation*

***GPIO\_speed\_define***

- #define: ***GPIO\_SPEED\_LOW ((uint32\_t)0x00000000)***  
*Low speed*
- #define: ***GPIO\_SPEED\_MEDIUM ((uint32\_t)0x00000001)***  
*Medium speed*
- #define: ***GPIO\_SPEED\_FAST ((uint32\_t)0x00000002)***  
*Fast speed*
- #define: ***GPIO\_SPEED\_HIGH ((uint32\_t)0x00000003)***  
*High speed*

## 20 HAL HASH Generic Driver

### 20.1 HASH Firmware driver registers structures

#### 20.1.1 HASH\_HandleTypeDef

*HASH\_HandleTypeDef* is defined in the `stm32f4xx_hal_hash.h`

##### Data Fields

- *HASH\_InitTypeDef Init*
- *uint8\_t \* pHashInBuffPtr*
- *uint8\_t \* pHashOutBuffPtr*
- *\_\_IO uint32\_t HashBuffSize*
- *\_\_IO uint32\_t HashInCount*
- *\_\_IO uint32\_t HashITCounter*
- *HAL\_StatusTypeDef Status*
- *HAL\_HASHPhaseTypeDef Phase*
- *DMA\_HandleTypeDef \* hdmain*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_HASH\_STATETypeDef State*

##### Field Documentation

- ***HASH\_InitTypeDef HASH\_HandleTypeDef::Init***
  - HASH required parameters
- ***uint8\_t\* HASH\_HandleTypeDef::pHashInBuffPtr***
  - Pointer to input buffer
- ***uint8\_t\* HASH\_HandleTypeDef::pHashOutBuffPtr***
  - Pointer to output buffer
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::HashBuffSize***
  - Size of buffer to be processed
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::HashInCount***
  - Counter of inputted data
- ***\_\_IO uint32\_t HASH\_HandleTypeDef::HashITCounter***
  - Counter of issued interrupts
- ***HAL\_StatusTypeDef HASH\_HandleTypeDef::Status***
  - HASH peripheral status
- ***HAL\_HASHPhaseTypeDef HASH\_HandleTypeDef::Phase***
  - HASH peripheral phase
- ***DMA\_HandleTypeDef\* HASH\_HandleTypeDef::hdmain***
  - HASH In DMA handle parameters
- ***HAL\_LockTypeDef HASH\_HandleTypeDef::Lock***
  - HASH locking object
- ***\_\_IO HAL\_HASH\_STATETypeDef HASH\_HandleTypeDef::State***
  - HASH peripheral state

## 20.1.2 HASH\_InitTypeDef

*HASH\_InitTypeDef* is defined in the stm32f4xx\_hal\_hash.h

### Data Fields

- *uint32\_t DataType*
- *uint32\_t KeySize*
- *uint8\_t \* pKey*

### Field Documentation

- *uint32\_t HASH\_InitTypeDef::DataType*
  - 32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of [HASH\\_Data\\_Type](#)
- *uint32\_t HASH\_InitTypeDef::KeySize*
  - The key size is used only in HMAC operation
- *uint8\_t\* HASH\_InitTypeDef::pKey*
  - The key is used only in HMAC operation

## 20.1.3 HASH\_DIGEST\_TypeDef

*HASH\_DIGEST\_TypeDef* is defined in the stm32f439xx.h

### Data Fields

- *\_\_IO uint32\_t HR*

### Field Documentation

- *\_\_IO uint32\_t HASH\_DIGEST\_TypeDef::HR[8]*
  - HASH digest registers, Address offset: 0x310-0x32C

## 20.1.4 HASH\_TypeDef

*HASH\_TypeDef* is defined in the stm32f439xx.h

### Data Fields

- *\_\_IO uint32\_t CR*
- *\_\_IO uint32\_t DIN*
- *\_\_IO uint32\_t STR*
- *\_\_IO uint32\_t HR*
- *\_\_IO uint32\_t IMR*
- *\_\_IO uint32\_t SR*
- *uint32\_t RESERVED*
- *\_\_IO uint32\_t CSR*

## Field Documentation

- `__IO uint32_t HASH_TypeDef::CR`
  - HASH control register, Address offset: 0x00
- `__IO uint32_t HASH_TypeDef::DIN`
  - HASH data input register, Address offset: 0x04
- `__IO uint32_t HASH_TypeDef::STR`
  - HASH start register, Address offset: 0x08
- `__IO uint32_t HASH_TypeDef::HR[5]`
  - HASH digest registers, Address offset: 0x0C-0x1C
- `__IO uint32_t HASH_TypeDef::IMR`
  - HASH interrupt enable register, Address offset: 0x20
- `__IO uint32_t HASH_TypeDef::SR`
  - HASH status register, Address offset: 0x24
- `uint32_t HASH_TypeDef::RESERVED[52]`
  - Reserved, 0x28-0xF4
- `__IO uint32_t HASH_TypeDef::CSR[54]`
  - HASH context swap registers, Address offset: 0x0F8-0x1CC

## 20.2 HASH Firmware driver API description

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

### 20.2.1 How to use this driver

The HASH HAL driver can be used as follows:

1. Initialize the HASH low level resources by implementing the `HAL_HASH_MsInit()`:
  - a. Enable the HASH interface clock using `__HASH_CLK_ENABLE()`
  - b. In case of using processing APIs based on interrupts (e.g. `HAL_HMAC_SHA1_Start_IT()`)
    - Configure the HASH interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the HASH IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In HASH IRQ handler, call `HAL_HASH_IRQHandler()`
  - c. In case of using DMA to control data transfer (e.g. `HAL_HMAC_SHA1_Start_DMA()`)
    - Enable the DMAx interface clock using `__DMAx_CLK_ENABLE()`
    - Configure and enable one DMA stream one for managing data transfer from memory to peripheral (input stream). Managing data transfer from peripheral to memory can be performed only using CPU
    - Associate the initialized DMA handle to the HASH DMA handle using `__HAL_LINKDMA()`
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Stream using `HAL_NVIC_SetPriority()` and `HAL_NVIC_EnableIRQ()`
2. Initialize the HASH HAL using `HAL_HASH_Init()`. This function configures mainly:
  - a. The data type: 1-bit, 8-bit, 16-bit and 32-bit.
  - b. For HMAC, the encryption key.
  - c. For HMAC, the key size used for encryption.
3. Three processing functions are available:

- a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished e.g. HAL\_HASH\_SHA1\_Start()
  - b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt e.g. HAL\_HASH\_SHA1\_Start\_IT()
  - c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA e.g. HAL\_HASH\_SHA1\_Start\_DMA()
4. When the processing function is called at first time after HAL\_HASH\_Init() the HASH peripheral is initialized and processes the buffer in input. After that, the digest computation is started. When processing multi-buffer use the accumulate function to write the data in the peripheral without starting the digest computation. In last buffer use the start function to input the last buffer and start the digest computation.
- a. e.g. HAL\_HASH\_SHA1\_Accumulate() : write 1st data buffer in the peripheral without starting the digest computation
  - b. write (n-1)th data buffer in the peripheral without starting the digest computation
  - c. HAL\_HASH\_SHA1\_Start() : write (n)th data buffer in the peripheral and start the digest computation
5. In HMAC mode, there is no Accumulate API. Only Start API is available.
6. In case of using DMA, call the DMA start processing e.g. HAL\_HASH\_SHA1\_Start\_DMA(). After that, call the finish function in order to get the digest value e.g. HAL\_HASH\_SHA1\_Finish()
7. Call HAL\_HASH\_DelInit() to deinitialize the HASH peripheral.

## 20.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the HASH according to the specified parameters in the HASH\_InitTypeDef and creates the associated handle.
- Deinitialize the HASH peripheral.
- Initialize the HASH MSP.
- Deinitialize HASH MSP.
- [\*\*HAL\\_HASH\\_Init\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_MspDelInit\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_InCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_ErrorCallback\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_DgstCpltCallback\(\)\*\*](#)

## 20.2.3 HASH processing using polling mode functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- MD5
- SHA1
- [\*\*HAL\\_HASH\\_MD5\\_Start\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_MD5\\_Accumulate\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_SHA1\\_Start\(\)\*\*](#)
- [\*\*HAL\\_HASH\\_SHA1\\_Accumulate\(\)\*\*](#)

## 20.2.4 HASH processing using interrupt mode functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- MD5
- SHA1
- `HAL_HASH_MD5_Start_IT()`
- `HAL_HASH_SHA1_Start_IT()`
- `HAL_HASH_IRQHandler()`

## 20.2.5 HASH processing using DMA mode functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- MD5
- SHA1
- `HAL_HASH_MD5_Start_DMA()`
- `HAL_HASH_MD5_Finish()`
- `HAL_HASH_SHA1_Start_DMA()`
- `HAL_HASH_SHA1_Finish()`

## 20.2.6 HMAC processing using polling mode functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- MD5
- SHA1
- `HAL_HMAC_MD5_Start()`
- `HAL_HMAC_SHA1_Start()`

## 20.2.7 HMAC processing using DMA mode functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- MD5
- SHA1
- `HAL_HMAC_MD5_Start_DMA()`
- `HAL_HMAC_SHA1_Start_DMA()`

## 20.2.8 Peripheral State functions

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

- `HAL_HASH_GetState()`

## 20.2.9 Initialization and de-initialization functions

### 20.2.9.1 HAL\_HASH\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HASH_Init ( <i>HASH_HandleTypeDef</i> * hhash)</b>
Function Description	Initializes the HASH according to the specified parameters in the <i>HASH_HandleTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a <i>HASH_HandleTypeDef</i> structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.9.2 HAL\_HASH\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_HASH_DeInit ( <i>HASH_HandleTypeDef</i> * hhash)</b>
Function Description	Deinitializes the HASH peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a <i>HASH_HandleTypeDef</i> structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before starting a new processing.</li> </ul>

### 20.2.9.3 HAL\_HASH\_MspInit

Function Name	<b>void HAL_HASH_MspInit ( <i>HASH_HandleTypeDef</i> * hhash)</b>
Function Description	Initializes the HASH MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a <i>HASH_HandleTypeDef</i> structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 20.2.9.4 HAL\_HASH\_MspDeInit

Function Name	<b>void HAL_HASH_MspDeInit ( <i>HASH_HandleTypeDef</i> * hhash)</b>
Function Description	Deinitializes HASH MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 20.2.9.5 HAL\_HASH\_InCpltCallback

Function Name	<b>void HAL_HASH_InCpltCallback ( <i>HASH_HandleTypeDef</i> * hhash)</b>
Function Description	Input data transfer complete callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 20.2.9.6 HAL\_HASH\_ErrorCallback

Function Name	<b>void HAL_HASH_ErrorCallback ( <i>HASH_HandleTypeDef</i> * hhash)</b>
Function Description	Data transfer Error callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

## Notes

- None.

**20.2.9.7 HAL\_HASH\_DgstCpltCallback**

Function Name	<b>void HAL_HASH_DgstCpltCallback ( <i>HASH_HandleTypeDef</i> * hhash)</b>
Function Description	Digest computation complete callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This callback is not relevant with DMA.</li> </ul>

**20.2.10 HASH processing functions using polling mode****20.2.10.1 HAL\_HASH\_MD5\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Start ( <i>HASH_HandleTypeDef</i> * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</b>
Function Description	Initializes the HASH peripheral in MD5 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is multiple of 64 bytes, appending the input buffer is possible. If the Size is not multiple of 64 bytes, the padding is managed by hardware and appending the input buffer is no more possible.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 16 bytes.</li> <li>• <b>Timeout</b> : Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.10.2 HAL\_HASH\_MD5\_Accumulate

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Accumulate (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</b> <b>Size)</b>
Function Description	Initializes the HASH peripheral in MD5 mode then writes the pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is multiple of 64 bytes, appending the input buffer is possible. If the Size is not multiple of 64 bytes, the padding is managed by hardware and appending the input buffer is no more possible.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.10.3 HAL\_HASH\_SHA1\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</b> <b>Size, uint8_t * pOutBuffer, uint32_t Timeout)</b>
Function Description	Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li> <li>• <b>Timeout</b> : Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.10.4 HAL\_HASH\_SHA1\_Accumulate

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Accumulate ( HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 20.2.11 HASH processing functions using interrupt mode

### 20.2.11.1 HAL\_HASH\_MD5\_Start\_IT

Function Name	<code>HAL_StatusTypeDef HAL_HASH_MD5_Start_IT ( HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)</code>
Function Description	Initializes the HASH peripheral in MD5 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pOutBuffer</b> : Pointer to the Output buffer (hashed buffer).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 16 bytes.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.11.2 HAL\_HASH\_SHA1\_Start\_IT

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Start_IT (</code> <code>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</code> <code>Size, uint8_t * pOutBuffer)</code>
Function Description	Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.11.3 HAL\_HASH\_IRQHandler

Function Name	<code>void HAL_HASH_IRQHandler ( HASH_HandleTypeDef * hhash )</code>
Function Description	This function handles HASH interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 20.2.12 HASH processing functions using DMA mode

### 20.2.12.1 HAL\_HASH\_MD5\_Start\_DMA

---

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</b> <b>Size)</b>
Function Description	Initializes the HASH peripheral in MD5 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.12.2 HAL\_HASH\_MD5\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Finish (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t</b> <b>Timeout)</b>
Function Description	Returns the computed digest in MD5 mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 16 bytes.</li> <li>• <b>Timeout</b> : Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 20.2.12.3 HAL\_HASH\_SHA1\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</b> <b>Size)</b>
Function Description	Initializes the HASH peripheral in SHA1 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that</li> </ul>

---

	contains the configuration information for HASH module
• <b>plnBuffer</b> : Pointer to the input buffer (buffer to be hashed).	
• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.	
Return values	• <b>HAL status</b>
Notes	• None.

#### 20.2.12.4 HAL\_HASH\_SHA1\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Finish (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t</b> <b>Timeout)</b>
Function Description	Returns the computed digest in SHA1 mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li> <li>• <b>Timeout</b> : Timeout value</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 20.2.13 HASH-MAC (HMAC) processing functions using polling mode

#### 20.2.13.1 HAL\_HMAC\_MD5\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_MD5_Start (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * plnBuffer, uint32_t</b> <b>Size, uint8_t * pOutBuffer, uint32_t Timeout)</b>
Function Description	Initializes the HASH peripheral in HMAC MD5 mode then processes plnBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>plnBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must</li> </ul>

---

	be 20 bytes.
Return values	• <b>Timeout</b> : Timeout value
Notes	• <b>HAL status</b>
	• None.

### 20.2.13.2 HAL\_HMAC\_SHA1\_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMAC_SHA1_Start (</code> <code>    HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</code> <code>    Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA1 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li> <li>• <b>Timeout</b> : Timeout value</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

## 20.2.14 HASH-MAC (HMAC) processing functions using DMA mode

### 20.2.14.1 HAL\_HMAC\_MD5\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HMAC_MD5_Start_DMA (</code> <code>    HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</code> <code>    Size)</code>
Function Description	Initializes the HASH peripheral in HMAC MD5 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not</li> </ul>

multiple of 64 bytes, the padding is managed by hardware.

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>             |

### 20.2.14.2 HAL\_HMAC\_SHA1\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_SHA1_Start_DMA (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</b> <b>Size)</b>
Function Description	Initializes the HASH peripheral in HMAC SHA1 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li><li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li><li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 20.2.15 Peripheral State functions

### 20.2.15.1 HAL\_HASH\_GetState

Function Name	<b>HAL_HASH_STATETypeDef HAL_HASH_GetState (</b> <b>HASH_HandleTypeDef * hhash)</b>
Function Description	return the HASH state
Parameters	<ul style="list-style-type: none"><li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 20.3 HASH Firmware driver defines

### 20.3.1 HASH

HASH

*HASH\_Algorithm\_Mode*

- #define: **HASH\_AlgoMode\_HASH** ((*uint32\_t*)0x00000000)

*Algorithm is HASH*

- #define: **HASH\_AlgoMode\_HMAC HASH\_CR\_MODE**

*Algorithm is HMAC*

*HASH\_Algo\_Selection*

- #define: **HASH\_AlgoSelection\_SHA1** ((*uint32\_t*)0x0000)

*HASH function is SHA1*

- #define: **HASH\_AlgoSelection\_SHA224 HASH\_CR\_ALGO\_1**

*HASH function is SHA224*

- #define: **HASH\_AlgoSelection\_SHA256 HASH\_CR\_ALGO**

*HASH function is SHA256*

- #define: **HASH\_AlgoSelection\_MD5 HASH\_CR\_ALGO\_0**

*HASH function is MD5*

*HASH\_Data\_Type*

- #define: **HASH\_DATATYPE\_32B** ((*uint32\_t*)0x0000)

*32-bit data. No swapping*

- #define: **HASH\_DATATYPE\_16B HASH\_CR\_DATATYPE\_0**

*16-bit data. Each half word is swapped*

- #define: **HASH\_DATATYPE\_8B HASH\_CR\_DATATYPE\_1**

*8-bit data. All bytes are swapped*

- #define: **HASH\_DATATYPE\_1B HASH\_CR\_DATATYPE**

*1-bit data. In the word all bits are swapped*

***HASH\_flags\_definition***

- #define: **HASH\_FLAG\_DINIS HASH\_SR\_DINIS**

*16 locations are free in the DIN : A new block can be entered into the input buffer*

- #define: **HASH\_FLAG\_DCIS HASH\_SR\_DCIS**

*Digest calculation complete*

- #define: **HASH\_FLAG\_DMAS HASH\_SR\_DMAS**

*DMA interface is enabled (DMAE=1) or a transfer is ongoing*

- #define: **HASH\_FLAG\_BUSY HASH\_SR\_BUSY**

*The hash core is Busy : processing a block of data*

- #define: **HASH\_FLAG\_DINNE HASH\_CR\_DINNE**

*DIN not empty : The input buffer contains at least one word of data*

***HASH\_HMAC\_Long\_key\_only\_for\_HMAC\_mode***

- #define: **HASH\_HMACKeyType\_ShortKey ((uint32\_t)0x00000000)**

*HMAC Key is <= 64 bytes*

- #define: **HASH\_HMACKeyType\_LongKey HASH\_CR\_LKEY**

*HMAC Key is > 64 bytes*

***HASH\_interrupts\_definition***

- #define: **HASH\_IT\_DINI HASH\_IMR\_DINIM**

*A new block can be entered into the input buffer (DIN)*

- #define: **HASH\_IT\_DCI HASH\_IMR\_DCIM**

*Digest calculation complete*

## 21 HAL HASH Extension Driver

### 21.1 HASHEx Firmware driver API description

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

#### 21.1.1 How to use this driver

The HASH HAL driver can be used as follows:

1. Initialize the HASH low level resources by implementing the HAL\_HASH\_MspInit():
  - a. Enable the HASH interface clock using \_\_HASH\_CLK\_ENABLE()
  - b. In case of using processing APIs based on interrupts (e.g.  
HAL\_HMACEx\_SHA224\_Start())
    - Configure the HASH interrupt priority using HAL\_NVIC\_SetPriority()
    - Enable the HASH IRQ handler using HAL\_NVIC\_EnableIRQ()
    - In HASH IRQ handler, call HAL\_HASH\_IRQHandler()
  - c. In case of using DMA to control data transfer (e.g.  
HAL\_HMACEx\_SH224\_Start\_DMA())
    - Enable the DMAx interface clock using \_\_DMAx\_CLK\_ENABLE()
    - Configure and enable one DMA stream one for managing data transfer from memory to peripheral (input stream). Managing data transfer from peripheral to memory can be performed only using CPU
    - Associate the initialized DMA handle to the HASH DMA handle using \_\_HAL\_LINKDMA()
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Stream: HAL\_NVIC\_SetPriority() and HAL\_NVIC\_EnableIRQ()
2. Initialize the HASH HAL using HAL\_HASH\_Init(). This function configures mainly:
  - a. The data type: 1-bit, 8-bit, 16-bit and 32-bit.
  - b. For HMAC, the encryption key.
  - c. For HMAC, the key size used for encryption.
3. Three processing functions are available:
  - a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished e.g.  
HAL\_HASHEx\_SHA224\_Start()
  - b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt e.g. HAL\_HASHEx\_SHA224\_Start\_IT()
  - c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA e.g.  
HAL\_HASHEx\_SHA224\_Start\_DMA()
4. When the processing function is called at first time after HAL\_HASH\_Init() the HASH peripheral is initialized and processes the buffer in input. After that, the digest computation is started. When processing multi-buffer use the accumulate function to write the data in the peripheral without starting the digest computation. In last buffer use the start function to input the last buffer and start the digest computation.
  - a. e.g. HAL\_HASHEx\_SHA224\_Accumulate() : write 1st data buffer in the peripheral without starting the digest computation
  - b. write (n-1)th data buffer in the peripheral without starting the digest computation
  - c. HAL\_HASHEx\_SHA224\_Start() : write (n)th data buffer in the peripheral and start the digest computation
5. In HMAC mode, there is no Accumulate API. Only Start API is available.

6. In case of using DMA, call the DMA start processing e.g.  
`HAL_HASHEx_SHA224_Start_DMA()`. After that, call the finish function in order to get the digest value e.g. `HAL_HASHEx_SHA224_Finish()`
7. Call `HAL_HASH_DeInit()` to deinitialize the HASH peripheral.

### 21.1.2 HASH processing using polling mode functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- SHA224
- SHA256
- `HAL_HASHEx_SHA224_Start()`
- `HAL_HASHEx_SHA256_Start()`
- `HAL_HASHEx_SHA224_Accumulate()`
- `HAL_HASHEx_SHA256_Accumulate()`

### 21.1.3 HMAC processing using polling mode functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- SHA224
- SHA256
- `HAL_HMACEx_SHA224_Start()`
- `HAL_HMACEx_SHA256_Start()`

### 21.1.4 HASH processing using interrupt functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- SHA224
- SHA256
- `HAL_HASHEx_SHA224_Start_IT()`
- `HAL_HASHEx_SHA256_Start_IT()`
- `HAL_HASHEx_IRQHandler()`

### 21.1.5 HASH processing using DMA functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- SHA224
- SHA256
- `HAL_HASHEx_SHA224_Start_DMA()`
- `HAL_HASHEx_SHA224_Finish()`
- `HAL_HASHEx_SHA256_Start_DMA()`
- `HAL_HASHEx_SHA256_Finish()`

## 21.1.6 HMAC processing using DMA functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- SHA224
- SHA256
- [\*\*\*HAL\\_HMACEEx\\_SHA224\\_Start\\_DMA\(\)\*\*\*](#)
- [\*\*\*HAL\\_HMACEEx\\_SHA256\\_Start\\_DMA\(\)\*\*\*](#)

## 21.1.7 HASH processing functions

### 21.1.7.1 HAL\_HASHEx\_SHA224\_Start

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start (     <b><i>HASH_HandleTypeDef</i></b> * hhash, uint8_t * pInBuffer, uint32_t     Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 28 bytes.</li> <li>• <b>Timeout</b> : Specify Timeout value</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 21.1.7.2 HAL\_HASHEx\_SHA256\_Start

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start (     <b><i>HASH_HandleTypeDef</i></b> * hhash, uint8_t * pInBuffer, uint32_t     Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not</li> </ul>

---

	multiple of 64 bytes, the padding is managed by hardware.
• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 32 bytes.	
• <b>Timeout</b> : Specify Timeout value	
Return values	• <b>HAL status</b>

Notes

- None.

### 21.1.7.3 HAL\_HASHEx\_SHA224\_Accumulate

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Accumulate (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</b> <b>Size)</b>
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li><b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li><b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li><b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	• <b>HAL status</b>
Notes	None.

### 21.1.7.4 HAL\_HASHEx\_SHA256\_Accumulate

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Accumulate (</b> <b>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</b> <b>Size)</b>
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li><b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li><b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li><b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	• <b>HAL status</b>

## Notes

- None.

## 21.1.8 HMAC processing functions using polling mode

### 21.1.8.1 HAL\_HMACEx\_SHA224\_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA224_Start (</code> <code>  HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</code> <code>  Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 21.1.8.2 HAL\_HMACEx\_SHA256\_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA256_Start (</code> <code>  HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</code> <code>  Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

## 21.1.9 HASH processing functions using interrupt mode

### 21.1.9.1 HAL\_HASHEx\_SHA224\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT (</b> <b><i>HASH_HandleTypeDef</i> * hhash, uint8_t * pInBuffer, uint32_t</b> <b><i>Size, uint8_t * pOutBuffer</i>)</b>
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"><li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li><li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li><li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li><li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 21.1.9.2 HAL\_HASHEx\_SHA256\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_IT (</b> <b><i>HASH_HandleTypeDef</i> * hhash, uint8_t * pInBuffer, uint32_t</b> <b><i>Size, uint8_t * pOutBuffer</i>)</b>
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"><li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li><li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li><li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li><li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 20 bytes.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>

## Notes

- None.

### 21.1.9.3 HAL\_HASHEx\_IRQHandler

Function Name	<b>void HAL_HASHEx_IRQHandler ( <i>HASH_HandleTypeDef</i> * hhash)</b>
Function Description	This function handles HASH interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 21.1.10 HASH processing functions using DMA mode

### 21.1.10.1 HAL\_HASHEx\_SHA224\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA ( <i>HASH_HandleTypeDef</i> * hhash, uint8_t * pInBuffer, uint32_t Size)</b>
Function Description	Initializes the HASH peripheral in SHA224 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 21.1.10.2 HAL\_HASHEx\_SHA224\_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish (</code> <code>HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t</code> <code>Timeout)</code>
Function Description	Returns the computed digest in SHA224.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 28 bytes.</li> <li>• <b>Timeout</b> : Timeout value</li> </ul>
Return values	• <b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 21.1.10.3 HAL\_HASHEx\_SHA256\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA (</code> <code>HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t</code> <code>Size)</code>
Function Description	Initializes the HASH peripheral in SHA256 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	• <b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 21.1.10.4 HAL\_HASHEx\_SHA256\_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (</code> <code>HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t</code> <code>Timeout)</code>
Function Description	Returns the computed digest in SHA256.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b> : pointer to a HASH_HandleTypeDef structure that</li> </ul>

---

	contains the configuration information for HASH module
• <b>pOutBuffer</b> : Pointer to the computed digest. Its size must be 32 bytes.	
• <b>Timeout</b> : Timeout value	
Return values	• <b>HAL status</b>

Notes

- None.

## 21.1.11 HMAC processing functions using DMA mode

### 21.1.11.1 HAL\_HMACEx\_SHA224\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_DMA (</code> <code>  <b>HASH_HandleTypeDef</b> * hhash, uint8_t * pInBuffer, uint32_t</code> <code>  Size)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA224 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li><b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li><b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 21.1.11.2 HAL\_HMACEx\_SHA256\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_DMA (</code> <code>  <b>HASH_HandleTypeDef</b> * hhash, uint8_t * pInBuffer, uint32_t</code> <code>  Size)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA256 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hhash</b> : pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li><b>pInBuffer</b> : Pointer to the input buffer (buffer to be hashed).</li> <li><b>Size</b> : Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>             |

## 21.2 HASHEx Firmware driver defines

### 21.2.1 HASHEx

HASHEx

## 22 HAL HCD Generic Driver

### 22.1 HCD Firmware driver registers structures

#### 22.1.1 HCD\_HandleTypeDef

*HCD\_HandleTypeDef* is defined in the `stm32f4xx_hal_hcd.h`

##### Data Fields

- *HCD\_TypeDef \* Instance*
- *HCD\_InitTypeDef Init*
- *HCD\_HCTypedef hc*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HCD\_StateTypeDef State*
- *void \* pData*

##### Field Documentation

- *HCD\_TypeDef\* HCD\_HandleTypeDef::Instance*
  - Register base address
- *HCD\_InitTypeDef HCD\_HandleTypeDef::Init*
  - HCD required parameters
- *HCD\_HCTypedef HCD\_HandleTypeDef::hc[15]*
  - Host channels parameters
- *HAL\_LockTypeDef HCD\_HandleTypeDef::Lock*
  - HCD peripheral status
- *\_\_IO HCD\_StateTypeDef HCD\_HandleTypeDef::State*
  - HCD communication state
- *void\* HCD\_HandleTypeDef::pData*
  - Pointer Stack Handler

### 22.2 HCD Firmware driver API description

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

#### 22.2.1 How to use this driver

1. Declare a `HCD_HandleTypeDef` handle structure, for example: `HCD_HandleTypeDef hhcd;`
2. Fill parameters of `Init` structure in `HCD` handle
3. Call `HAL_HCD_Init()` API to initialize the `HCD` peripheral (Core, Host core, ...)
4. Initialize the `HCD` low level resources through the `HAL_HCD_MspInit()` API:
  - a. Enable the `HCD/USB` Low Level interface clock using the following macros
    - `__OTGFS-OTG_CLK_ENABLE()` or `__OTGHS-OTG_CLK_ENABLE()`

- `__OTGHSULPI_CLK_ENABLE()` For High Speed Mode
- b. Initialize the related GPIO clocks
- c. Configure HCD pin-out
- d. Configure HCD NVIC interrupt
- 5. Associate the Upper USB Host stack to the HAL HCD Driver:
  - a. `hhcd.pData = phost;`
- 6. Enable HCD transmission and reception:
  - a. `HAL_HCD_Start();`

## 22.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- `HAL_HCD_Init()`
- `HAL_HCD_HC_Init()`
- `HAL_HCD_HC_Halt()`
- `HAL_HCD_DelInit()`
- `HAL_HCD_MspInit()`
- `HAL_HCD_MspDelInit()`

## 22.2.3 IO operation functions

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

## 22.2.4 Peripheral Control functions

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

- `HAL_HCD_Start()`
- `HAL_HCD_Stop()`
- `HAL_HCD_ResetPort()`

## 22.2.5 Peripheral State functions

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

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

## 22.2.6 Initialization and de-initialization functions

### 22.2.6.1 HAL\_HCD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HCD_Init ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	Initialize the host driver.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 22.2.6.2 HAL\_HCD\_HC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_Init ( <i>HCD_HandleTypeDef</i> * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)</b>
Function Description	Initialize a host channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> <li>• <b>ch_num</b> : Channel number. This parameter can be a value from 1 to 15</li> <li>• <b>epnum</b> : Endpoint number. This parameter can be a value from 1 to 15</li> <li>• <b>dev_address</b> : Current device address This parameter can be a value from 0 to 255</li> <li>• <b>speed</b> : Current device speed. This parameter can be one of these values: HCD_SPEED_HIGH: High speed mode, HCD_SPEED_FULL: Full speed mode, HCD_SPEED_LOW: Low speed mode</li> <li>• <b>ep_type</b> : Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type, EP_TYPE_ISOC: Isochronous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type</li> <li>• <b>mps</b> : Max Packet Size. This parameter can be a value from 0 to 32K</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 22.2.6.3 HAL\_HCD\_HC\_Halt

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_Halt ( <i>HCD_HandleTypeDef</i> * hhcd, uint8_t ch_num)</b>
Function Description	Halt a host channel.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li><li>• <b>ch_num</b> : Channel number. This parameter can be a value from 1 to 15</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.6.4 HAL\_HCD\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_HCD_DelInit ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	Delinitialize the host driver.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>

### 22.2.6.5 HAL\_HCD\_MsplInit

Function Name	<b>void HAL_HCD_MsplInit ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	Initializes the HCD MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 22.2.6.6 HAL\_HCD\_MspDelInit

Function Name	<b>void HAL_HCD_MspDelInit ( <i>HCD_HandleTypeDefDef</i> * hhcd)</b>
Function Description	Deinitializes HCD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 22.2.7 IO operation functions

### 22.2.7.1 HAL\_HCD\_HC\_SubmitRequest

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest ( <i>HCD_HandleTypeDefDef</i> * hhcd, uint8_t pipe, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)</b>
Function Description	Submit a new URB for processing.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> <li>• <b>ch_num</b> : Channel number. This parameter can be a value from 1 to 15</li> <li>• <b>direction</b> : Channel number. This parameter can be one of these values: 0 : Output / 1 : Input</li> <li>• <b>ep_type</b> : Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type/ EP_TYPE_ISOC: Isochronous type/ EP_TYPE_BULK: Bulk type/ EP_TYPE_INTR: Interrupt type/</li> <li>• <b>token</b> : Endpoint Type. This parameter can be one of these values: 0: HC_PID_SETUP / 1: HC_PID_DATA1</li> <li>• <b>pbuff</b> : pointer to URB data</li> <li>• <b>length</b> : Length of URB data</li> <li>• <b>do_ping</b> : activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 22.2.7.2 HAL\_HCD\_IRQHandler

Function Name	<b>void HAL_HCD_IRQHandler ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	This function handles HCD interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.7.3 HAL\_HCD\_SOF\_Callback

Function Name	<b>void HAL_HCD_SOF_Callback ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	SOF callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.7.4 HAL\_HCD\_Connect\_Callback

Function Name	<b>void HAL_HCD_Connect_Callback ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	Connexion Event callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.7.5 HAL\_HCD\_Disconnect\_Callback

Function Name	<code>void HAL_HCD_Disconnect_Callback ( <i>HCD_HandleTypeDef</i> * hhcd)</code>
Function Description	Disconnection Event callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 22.2.7.6 HAL\_HCD\_HC\_NotifyURBChange\_Callback

Function Name	<code>void HAL_HCD_HC_NotifyURBChange_Callback ( <i>HCD_HandleTypeDef</i> * hhcd, uint8_t chnum, <i>HCD_URBStateTypeDef</i> urb_state)</code>
Function Description	Notify URB state change callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> <li>• <b>chnum</b> : Channel number. This parameter can be a value from 1 to 15</li> <li>• <b>urb_state</b> : This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 22.2.8 Peripheral Control functions

### 22.2.8.1 HAL\_HCD\_Start

Function Name	<code>HAL_StatusTypeDef HAL_HCD_Start ( <i>HCD_HandleTypeDef</i> * hhcd)</code>
Function Description	Start the host driver.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 22.2.8.2 HAL\_HCD\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_HCD_Stop ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	Stop the host driver.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 22.2.8.3 HAL\_HCD\_ResetPort

Function Name	<b>HAL_StatusTypeDef HAL_HCD_ResetPort ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	Reset the host port.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 22.2.9 Peripheral State functions

### 22.2.9.1 HAL\_HCD\_GetState

Function Name	<b>HCD_StateTypeDef HAL_HCD_GetState ( <i>HCD_HandleTypeDef</i> * hhcd)</b>
Function Description	Return the HCD state.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b> : HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>

## Notes

- None.

**22.2.9.2 HAL\_HCD\_HC\_GetURBState**

Function Name	<b>HCD_URBStateTypeDef HAL_HCD_HC_GetURBState ( <i>HCD_HandleTypeDef</i> * hhcd, uint8_t chnum)</b>
Function Description	Return URB state for a channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> <li>• <b>chnum</b> : Channel number. This parameter can be a value from 1 to 15</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>URB state. This parameter can be one of these values:</b> <b>URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**22.2.9.3 HAL\_HCD\_HC\_GetXferCount**

Function Name	<b>uint32_t HAL_HCD_HC_GetXferCount ( <i>HCD_HandleTypeDef</i> * hhcd, uint8_t chnum)</b>
Function Description	Return the last host transfer size.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> <li>• <b>chnum</b> : Channel number. This parameter can be a value from 1 to 15</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>last transfer size in byte</b></li> </ul>

## Notes

- None.

**22.2.9.4 HAL\_HCD\_HC\_GetState**

Function Name	<b>HCD_HCStateTypeDef HAL_HCD_HC_GetState (</b>
---------------	---

***HCD\_HandleTypeDefDef \* hhcd, uint8\_t chnum)***

Function Description	Return the Host Channel state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> <li>• <b>chnum</b> : Channel number. This parameter can be a value from 1 to 15</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Host channel state</b> This parameter can be one of the these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR/</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**22.2.9.5 HAL\_HCD\_GetCurrentFrame**

Function Name	<b>uint32_t HAL_HCD_GetCurrentFrame ( <i>HCD_HandleTypeDefDef * hhcd</i>)</b>
Function Description	Return the current Host frame number.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Current Host frame number</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**22.2.9.6 HAL\_HCD\_GetCurrentSpeed**

Function Name	<b>uint32_t HAL_HCD_GetCurrentSpeed ( <i>HCD_HandleTypeDefDef * hhcd</i>)</b>
Function Description	Return the Host enumeration speed.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b> : HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Enumeration speed</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 22.3 HCD Firmware driver defines

### 22.3.1 HCD

HCD

*HCD\_PHY\_Module*

- #define: ***HCD\_PHY\_ULPI*** 1
  
- #define: ***HCD\_PHY\_EMBEDDED*** 2

*HCD\_Speed*

- #define: ***HCD\_SPEED\_HIGH*** 0
  
- #define: ***HCD\_SPEED\_LOW*** 2
  
- #define: ***HCD\_SPEED\_FULL*** 3

## 23 HAL I2C Generic Driver

### 23.1 I2C Firmware driver registers structures

#### 23.1.1 I2C\_HandleTypeDef

*I2C\_HandleTypeDef* is defined in the `stm32f4xx_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*

#### 23.1.2 I2C\_InitTypeDef

*I2C\_InitTypeDef* is defined in the `stm32f4xx_hal_i2c.h`

##### Data Fields

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

### Field Documentation

- `uint32_t I2C_InitTypeDef::ClockSpeed`
  - Specifies the clock frequency. This parameter must be set to a value lower than 400kHz
- `uint32_t I2C_InitTypeDef::DutyCycle`
  - Specifies the I2C fast mode duty cycle. This parameter can be a value of `I2C_duty_cycle_in_fast_mode`
- `uint32_t I2C_InitTypeDef::OwnAddress1`
  - Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- `uint32_t I2C_InitTypeDef::AddressingMode`
  - Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of `I2C_addressing_mode`
- `uint32_t I2C_InitTypeDef::DualAddressMode`
  - Specifies if dual addressing mode is selected. This parameter can be a value of `I2C_dual_addressing_mode`
- `uint32_t I2C_InitTypeDef::OwnAddress2`
  - Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- `uint32_t I2C_InitTypeDef::GeneralCallMode`
  - Specifies if general call mode is selected. This parameter can be a value of `I2C_general_call_addressing_mode`
- `uint32_t I2C_InitTypeDef::NoStretchMode`
  - Specifies if nostretch mode is selected. This parameter can be a value of `I2C_nostretch_mode`

### 23.1.3 I2C\_TypeDef

`I2C_TypeDef` is defined in the `stm32f439xx.h`

#### Data Fields

- `__IO uint32_t CR1`
- `__IO uint32_t CR2`
- `__IO uint32_t OAR1`
- `__IO uint32_t OAR2`
- `__IO uint32_t DR`
- `__IO uint32_t SR1`

- `__IO uint32_t SR2`
- `__IO uint32_t CCR`
- `__IO uint32_t TRISE`
- `__IO uint32_t FLTR`

### Field Documentation

- `__IO uint32_t I2C_TypeDef::CR1`
  - I2C Control register 1, Address offset: 0x00
- `__IO uint32_t I2C_TypeDef::CR2`
  - I2C Control register 2, Address offset: 0x04
- `__IO uint32_t I2C_TypeDef::OAR1`
  - I2C Own address register 1, Address offset: 0x08
- `__IO uint32_t I2C_TypeDef::OAR2`
  - I2C Own address register 2, Address offset: 0x0C
- `__IO uint32_t I2C_TypeDef::DR`
  - I2C Data register, Address offset: 0x10
- `__IO uint32_t I2C_TypeDef::SR1`
  - I2C Status register 1, Address offset: 0x14
- `__IO uint32_t I2C_TypeDef::SR2`
  - I2C Status register 2, Address offset: 0x18
- `__IO uint32_t I2C_TypeDef::CCR`
  - I2C Clock control register, Address offset: 0x1C
- `__IO uint32_t I2C_TypeDef::TRISE`
  - I2C TRISE register, Address offset: 0x20
- `__IO uint32_t I2C_TypeDef::FLTR`
  - I2C FLTR register, Address offset: 0x24

## 23.2 I2C Firmware driver API description

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

### 23.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C\_HandleTypeDef handle structure, for example: I2C\_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implement the HAL\_I2C\_MspInit() API:
  - a. Enable the I2Cx interface clock
  - b. I2C pins configuration
    - Enable the clock for the I2C GPIOs
    - Configure I2C pins as alternate function open-drain
  - c. NVIC configuration if you need to use interrupt process
    - Configure the I2Cx interrupt priority
    - Enable the NVIC I2C IRQ Channel
  - d. DMA Configuration if you need to use DMA process
    - Declare a DMA\_HandleTypeDef handle structure for the transmit or receive stream
    - Enable the DMAx interface clock using

- Configure the DMA handle parameters
  - Configure the DMA Tx or Rx Stream
  - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream
3. Configure the Communication Speed, Duty cycle, Addressing mode, Own Address1, Dual Addressing mode, Own Address2, General call and Nostretch mode in the hi2c Init structure.
  4. Initialize the I2C registers by calling the HAL\_I2C\_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL\_I2C\_MspInit(&hi2c) API.
  5. To check if target device is ready for communication, use the function HAL\_I2C\_IsDeviceReady()
  6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

### **Polling mode IO operation**

- Transmit in master mode an amount of data in blocking mode using HAL\_I2C\_Master\_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL\_I2C\_Master\_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL\_I2C\_Slave\_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL\_I2C\_Slave\_Receive()

### **Polling mode IO MEM operation**

- Write an amount of data in blocking mode to a specific memory address using HAL\_I2C\_Mem\_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL\_I2C\_Mem\_Read()

### **Interrupt mode IO operation**

- Transmit in master mode an amount of data in non blocking mode using HAL\_I2C\_Master\_Transmit\_IT()
- At transmission end of transfer HAL\_I2C\_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode using HAL\_I2C\_Master\_Receive\_IT()
- At reception end of transfer HAL\_I2C\_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode using HAL\_I2C\_Slave\_Transmit\_IT()
- At transmission end of transfer HAL\_I2C\_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveTxCpltCallback

- Receive in slave mode an amount of data in non blocking mode using HAL\_I2C\_Slave\_Receive\_IT()
- At reception end of transfer HAL\_I2C\_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### 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 MEM end of write transfer HAL\_I2C\_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemTxCpltCallback
- Read an amount of data in no-blocking mode with Interrupt from a specific memory address using HAL\_I2C\_Mem\_Read\_IT()
- At MEM end of read transfer HAL\_I2C\_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### DMA mode IO operation

- Transmit in master mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Master\_Transmit\_DMA()
- At transmission end of transfer HAL\_I2C\_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Master\_Receive\_DMA()
- At reception end of transfer HAL\_I2C\_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Slave\_Transmit\_DMA()
- At transmission end of transfer HAL\_I2C\_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode (DMA) using HAL\_I2C\_Slave\_Receive\_DMA()
- At reception end of transfer HAL\_I2C\_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### DMA mode IO MEM operation

- Write an amount of data in no-blocking mode with DMA to a specific memory address using HAL\_I2C\_Mem\_Write\_DMA()
- At MEM end of write transfer HAL\_I2C\_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemTxCpltCallback

- Read an amount of data in no-blocking mode with DMA from a specific memory address using HAL\_I2C\_Mem\_Read\_DMA()
- At MEM end of read transfer HAL\_I2C\_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MemRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### I2C HAL driver macros list

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 : Clear the specified I2C pending flag
- \_\_HAL\_I2C\_ENABLE\_IT: Enable the specified I2C interrupt
- \_\_HAL\_I2C\_DISABLE\_IT: Disable the specified I2C interrupt



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

### 23.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Cx peripheral:

- User must Implement HAL\_I2C\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL\_I2C\_Init() to configure the selected device with the selected configuration:
  - Communication Speed
  - Duty cycle
  - Addressing mode
  - Own Address 1
  - Dual Addressing mode
  - Own Address 2
  - General call mode
  - Nostretch mode
- Call the function HAL\_I2C\_DeInit() to restore the default configuration of the selected I2Cx peripheral.
- **HAL\_I2C\_Init()**
- **HAL\_I2C\_DeInit()**
- **HAL\_I2C\_MspInit()**
- **HAL\_I2C\_MspDeInit()**

### 23.2.3 IO operation functions

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

1. There are two modes of transfer:

- Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
- HAL\_I2C\_Master\_Transmit()
  - HAL\_I2C\_Master\_Receive()
  - HAL\_I2C\_Slave\_Transmit()
  - HAL\_I2C\_Slave\_Receive()
  - HAL\_I2C\_Mem\_Write()
  - HAL\_I2C\_Mem\_Read()
  - HAL\_I2C\_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
- HAL\_I2C\_Master\_Transmit\_IT()
  - HAL\_I2C\_Master\_Receive\_IT()
  - HAL\_I2C\_Slave\_Transmit\_IT()
  - HAL\_I2C\_Slave\_Receive\_IT()
  - HAL\_I2C\_Mem\_Write\_IT()
  - HAL\_I2C\_Mem\_Read\_IT()
4. No-Blocking mode functions with DMA are :
- HAL\_I2C\_Master\_Transmit\_DMA()
  - HAL\_I2C\_Master\_Receive\_DMA()
  - HAL\_I2C\_Slave\_Transmit\_DMA()
  - HAL\_I2C\_Slave\_Receive\_DMA()
  - HAL\_I2C\_Mem\_Write\_DMA()
  - HAL\_I2C\_Mem\_Read\_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
- HAL\_I2C\_MemTxCpltCallback()
  - HAL\_I2C\_MemRxCpltCallback()
  - HAL\_I2C\_MasterTxCpltCallback()
  - HAL\_I2C\_MasterRxCpltCallback()
  - HAL\_I2C\_SlaveTxCpltCallback()
  - HAL\_I2C\_SlaveRxCpltCallback()
  - HAL\_I2C\_ErrorCallback()
- ***HAL\_I2C\_Master\_Transmit()***
  - ***HAL\_I2C\_Master\_Receive()***
  - ***HAL\_I2C\_Slave\_Transmit()***
  - ***HAL\_I2C\_Slave\_Receive()***
  - ***HAL\_I2C\_Master\_Transmit\_IT()***
  - ***HAL\_I2C\_Master\_Receive\_IT()***
  - ***HAL\_I2C\_Slave\_Transmit\_IT()***
  - ***HAL\_I2C\_Slave\_Receive\_IT()***
  - ***HAL\_I2C\_Master\_Transmit\_DMA()***
  - ***HAL\_I2C\_Master\_Receive\_DMA()***
  - ***HAL\_I2C\_Slave\_Transmit\_DMA()***
  - ***HAL\_I2C\_Slave\_Receive\_DMA()***
  - ***HAL\_I2C\_Mem\_Write()***
  - ***HAL\_I2C\_Mem\_Read()***
  - ***HAL\_I2C\_Mem\_Write\_IT()***
  - ***HAL\_I2C\_Mem\_Read\_IT()***

- [\*HAL\\_I2C\\_Mem\\_Write\\_DMA\(\)\*](#)
- [\*HAL\\_I2C\\_Mem\\_Read\\_DMA\(\)\*](#)
- [\*HAL\\_I2C\\_IsDeviceReady\(\)\*](#)
- [\*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\(\)\*](#)

### 23.2.4 Peripheral State and Errors functions

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

- [\*HAL\\_I2C\\_GetState\(\)\*](#)
- [\*HAL\\_I2C\\_GetError\(\)\*](#)

### 23.2.5 Initialization and de-initialization functions

#### 23.2.5.1 HAL\_I2C\_Init

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Init ( <i>I2C_HandleTypeDef</i> * <i>hi2c</i>)</b>
Function Description	Initializes the I2C according to the specified parameters in the <i>I2C_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for I2C module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 23.2.5.2 HAL\_I2C\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_I2C_DelInit ( <i>I2C_HandleTypeDef</i> * <i>hi2c</i>)</b>
Function Description	Deinitializes the I2C peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for I2C module</li> </ul>

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>             |

### 23.2.5.3 HAL\_I2C\_MspInit

Function Name	<b>void HAL_I2C_MspInit ( <i>I2C_HandleTypeDef</i> * hi2c)</b>
Function Description	I2C MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 23.2.5.4 HAL\_I2C\_MspDelInit

Function Name	<b>void HAL_I2C_MspDelInit ( <i>I2C_HandleTypeDef</i> * hi2c)</b>
Function Description	I2C MSP DelInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 23.2.6 IO operation functions

### 23.2.6.1 HAL\_I2C\_Master\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit ( <i>I2C_HandleTypeDef</i> * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
---------------	--

Function Description	Transmits in master mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li><b>DevAddress</b> : Target device address</li> <li><b>pData</b> : Pointer to data buffer</li> <li><b>Size</b> : Amount of data to be sent</li> <li><b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 23.2.6.2 HAL\_I2C\_Master\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive (</b> <b>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receives in master mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li><b>DevAddress</b> : Target device address</li> <li><b>pData</b> : Pointer to data buffer</li> <li><b>Size</b> : Amount of data to be sent</li> <li><b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 23.2.6.3 HAL\_I2C\_Slave\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit (</b> <b>I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmits in slave mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li><b>pData</b> : Pointer to data buffer</li> <li><b>Size</b> : Amount of data to be sent</li> </ul>

- |               |                                     |
|---------------|-------------------------------------|
| Return values | • <b>Timeout</b> : Timeout duration |
| Notes         | • <b>HAL status</b>                 |
|               | • None.                             |

#### 23.2.6.4 HAL\_I2C\_Slave\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive (</b> <b>I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size,</b> <b>uint32_t Timeout)</b>
Function Description	Receive in slave mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li><li>• <b>pData</b> : Pointer to data buffer</li><li>• <b>Size</b> : Amount of data to be sent</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 23.2.6.5 HAL\_I2C\_Master\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (</b> <b>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t *</b> <b>pData, uint16_t Size)</b>
Function Description	Transmit in master mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li><li>• <b>DevAddress</b> : Target device address</li><li>• <b>pData</b> : Pointer to data buffer</li><li>• <b>Size</b> : Amount of data to be sent</li></ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 23.2.6.6 HAL\_I2C\_Master\_Receive\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Master_Receive_IT ( I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</code>
Function Description	Receive in master mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.7 HAL\_I2C\_Slave\_Transmit\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT ( I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</code>
Function Description	Transmit in slave mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.8 HAL\_I2C\_Slave\_Receive\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (</code>
---------------	---

***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	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**23.2.6.9 HAL\_I2C\_Master\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (</b> <b><i>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</i></b>
Function Description	Transmit in master mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**23.2.6.10 HAL\_I2C\_Master\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (</b> <b><i>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</i></b>
Function Description	Receive in master mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> </ul>

---

	<ul style="list-style-type: none"> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.11 HAL\_I2C\_Slave\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (</b> <b>I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.12 HAL\_I2C\_Slave\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (</b> <b>I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.13 HAL\_I2C\_Mem\_Write

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Write an amount of data in blocking mode to a specific memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>MemAddress</b> : Internal memory address</li> <li>• <b>MemAddSize</b> : Size of internal memory address</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	• <b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.14 HAL\_I2C\_Mem\_Read

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Read an amount of data in blocking mode from a specific memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>MemAddress</b> : Internal memory address</li> <li>• <b>MemAddSize</b> : Size of internal memory address</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	• <b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.15 HAL\_I2C\_Mem\_Write\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (     <i>I2C_HandleTypeDef</i> * hi2c, uint16_t DevAddress, uint16_t     MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t     Size)</code>
Function Description	Write an amount of data in no-blocking mode with Interrupt to a specific memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>MemAddress</b> : Internal memory address</li> <li>• <b>MemAddSize</b> : Size of internal memory address</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.16 HAL\_I2C\_Mem\_Read\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (     <i>I2C_HandleTypeDef</i> * hi2c, uint16_t DevAddress, uint16_t     MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t     Size)</code>
Function Description	Read an amount of data in no-blocking mode with Interrupt from a specific memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>MemAddress</b> : Internal memory address</li> <li>• <b>MemAddSize</b> : Size of internal memory address</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.17 HAL\_I2C\_Mem\_Write\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size)</code>
Function Description	Write an amount of data in no-blocking mode with DMA to a specific memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>MemAddress</b> : Internal memory address</li> <li>• <b>MemAddSize</b> : Size of internal memory address</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.18 HAL\_I2C\_Mem\_Read\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t</code> <code>MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t</code> <code>Size)</code>
Function Description	Reads an amount of data in no-blocking mode with DMA from a specific memory address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>MemAddress</b> : Internal memory address</li> <li>• <b>MemAddSize</b> : Size of internal memory address</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be read</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.19 HAL\_I2C\_IsDeviceReady

Function Name	<code>HAL_StatusTypeDef HAL_I2C_IsDeviceReady (</code> <code>I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t</code> <code>Trials, uint32_t Timeout)</code>
Function Description	Checks if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> <li>• <b>DevAddress</b> : Target device address</li> <li>• <b>Trials</b> : Number of trials</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function is used with Memory devices</li> </ul>

### 23.2.6.20 HAL\_I2C\_EV\_IRQHandler

Function Name	<code>void HAL_I2C_EV_IRQHandler ( I2C_HandleTypeDef * hi2c)</code>
Function Description	This function handles I2C event interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 23.2.6.21 HAL\_I2C\_ER\_IRQHandler

Function Name	<code>void HAL_I2C_ER_IRQHandler ( I2C_HandleTypeDef * hi2c)</code>
Function Description	This function handles I2C error interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

- 
- |       |   |
|-------|---|
| Notes | <ul style="list-style-type: none"><li>None.</li></ul> |
|-------|---|

### 23.2.6.22 HAL\_I2C\_MasterTxCpltCallback

Function Name	<b>void HAL_I2C_MasterTxCpltCallback ( <i>I2C_HandleTypeDef</i> * <i>hi2c</i> )</b>
Function Description	Master Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li><b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 23.2.6.23 HAL\_I2C\_MasterRxCpltCallback

Function Name	<b>void HAL_I2C_MasterRxCpltCallback ( <i>I2C_HandleTypeDef</i> * <i>hi2c</i> )</b>
Function Description	Master Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li><b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 23.2.6.24 HAL\_I2C\_SlaveTxCpltCallback

Function Name	<b>void HAL_I2C_SlaveTxCpltCallback ( <i>I2C_HandleTypeDef</i> * <i>hi2c</i> )</b>
Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li><b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that</li></ul>

contains the configuration information for I2C module

Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 23.2.6.25 HAL\_I2C\_SlaveRxCpltCallback

Function Name	<b>void HAL_I2C_SlaveRxCpltCallback ( <i>I2C_HandleTypeDef</i> * <b>hi2c</b>)</b>
Function Description	Slave Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 23.2.6.26 HAL\_I2C\_MemTxCpltCallback

Function Name	<b>void HAL_I2C_MemTxCpltCallback ( <i>I2C_HandleTypeDef</i> * <b>hi2c</b>)</b>
Function Description	Memory Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 23.2.6.27 HAL\_I2C\_MemRxCpltCallback

Function Name	<b>void HAL_I2C_MemRxCpltCallback ( <i>I2C_HandleTypeDef</i> * <b>hi2c</b>)</b>
---------------	---

Function Description	Memory Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 23.2.6.28 HAL\_I2C\_ErrorCallback

Function Name	<b>void HAL_I2C_ErrorCallback ( <i>I2C_HandleTypeDef</i> * hi2c)</b>
Function Description	I2C error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 23.2.7 Peripheral State and Errors functions

### 23.2.7.1 HAL\_I2C\_GetState

Function Name	<b>HAL_I2C_StateTypeDef HAL_I2C_GetState ( <i>I2C_HandleTypeDef</i> * hi2c)</b>
Function Description	Returns the I2C state.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>

Notes

### 23.2.7.2 HAL\_I2C\_GetError

Function Name	<code>uint32_t HAL_I2C_GetError ( I2C_HandleTypeDef * hi2c)</code>
Function Description	Return the I2C error code.
Parameters	<ul style="list-style-type: none"><li>• <code>hi2c</code> : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>I2C Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 23.3 I2C Firmware driver defines

### 23.3.1 I2C

I2C

*I2C\_addressing\_mode*

- #define: `I2C_ADDRESSINGMODE_7BIT ((uint32_t)0x00004000)`
- #define: `I2C_ADDRESSINGMODE_10BIT (I2C_OAR1_ADDMODE | ((uint32_t)0x00004000))`

*I2C\_dual\_addressing\_mode*

- #define: `I2C_DUALADDRESS_DISABLED ((uint32_t)0x00000000)`
- #define: `I2C_DUALADDRESS_ENABLED I2C_OAR2_ENDUAL`

*I2C\_duty\_cycle\_in\_fast\_mode*

- #define: `I2C_DUTYCYCLE_2 ((uint32_t)0x00000000)`
- #define: `I2C_DUTYCYCLE_16_9 I2C_CCR_DUTY`

*I2C\_Flag\_definition*

- #define: *I2C\_FLAG\_SMBALERT* ((*uint32\_t*)0x00018000)
- #define: *I2C\_FLAG\_TIMEOUT* ((*uint32\_t*)0x00014000)
- #define: *I2C\_FLAG\_PECERR* ((*uint32\_t*)0x00011000)
- #define: *I2C\_FLAG\_OVR* ((*uint32\_t*)0x00010800)
- #define: *I2C\_FLAG\_AF* ((*uint32\_t*)0x00010400)
- #define: *I2C\_FLAG\_ARLO* ((*uint32\_t*)0x00010200)
- #define: *I2C\_FLAG\_BERR* ((*uint32\_t*)0x00010100)
- #define: *I2C\_FLAG\_TXE* ((*uint32\_t*)0x00010080)
- #define: *I2C\_FLAG\_RXNE* ((*uint32\_t*)0x00010040)
- #define: *I2C\_FLAG\_STOPF* ((*uint32\_t*)0x00010010)
- #define: *I2C\_FLAG\_ADD10* ((*uint32\_t*)0x00010008)
- #define: *I2C\_FLAG\_BTF* ((*uint32\_t*)0x00010004)

- #define: *I2C\_FLAG\_ADDR* ((*uint32\_t*)0x00010002)
- #define: *I2C\_FLAG\_SB* ((*uint32\_t*)0x00010001)
- #define: *I2C\_FLAG\_DUALF* ((*uint32\_t*)0x00100080)
- #define: *I2C\_FLAG\_SMBHOST* ((*uint32\_t*)0x00100040)
- #define: *I2C\_FLAG\_SMBDEFAULT* ((*uint32\_t*)0x00100020)
- #define: *I2C\_FLAG\_GENCALL* ((*uint32\_t*)0x00100010)
- #define: *I2C\_FLAG\_TRA* ((*uint32\_t*)0x00100004)
- #define: *I2C\_FLAG\_BUSY* ((*uint32\_t*)0x00100002)
- #define: *I2C\_FLAG\_MSL* ((*uint32\_t*)0x00100001)

#### *I2C\_general\_call\_addressing\_mode*

- #define: *I2C\_GENERALCALL\_DISABLED* ((*uint32\_t*)0x00000000)
- #define: *I2C\_GENERALCALL\_ENABLED* *I2C\_CR1\_ENGC*

#### *I2C Interrupt configuration definition*

- #define: *I2C\_IT\_BUF* *I2C\_CR2\_ITBUFEN*

- #define: *I2C\_IT\_EVT I2C\_CR2\_ITEVREN*

- #define: *I2C\_IT\_ERR I2C\_CR2\_ITERREN*

#### *I2C\_Memory\_Address\_Size*

- #define: *I2C\_MEMADD\_SIZE\_8BIT ((uint32\_t)0x00000001)*

- #define: *I2C\_MEMADD\_SIZE\_16BIT ((uint32\_t)0x00000010)*

#### *I2C\_nostretch\_mode*

- #define: *I2C\_NOSTRETCH\_DISABLED ((uint32\_t)0x00000000)*

- #define: *I2C\_NOSTRETCH\_ENABLED I2C\_CR1\_NOSTRETCH*

## 24 HAL I2C Extension Driver

### 24.1 I2CEEx Firmware driver API description

The following section lists the various functions of the I2CEEx library.

#### 24.1.1 I2C peripheral extension features

Comparing to other previous devices, the I2C interface for STM32F427xx/437xx/429xx/439xx devices contains the following additional features :

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter

#### 24.1.2 How to use this driver

This driver provides functions to configure Noise Filter

1. Configure I2C Analog noise filter using the function `HAL_I2C_AnalogFilter_Config()`
2. Configure I2C Digital noise filter using the function `HAL_I2C_DigitalFilter_Config()`

#### 24.1.3 Extension features functions

This section provides functions allowing to:

- Configure Noise Filters
- [`HAL\_I2CEEx\_AnalogFilter\_Config\(\)`](#)
- [`HAL\_I2CEEx\_DigitalFilter\_Config\(\)`](#)

#### 24.1.4 Extension features functions

##### 24.1.4.1 `HAL_I2CEEx_AnalogFilter_Config`

Function Name	<code>HAL_StatusTypeDef HAL_I2CEEx_AnalogFilter_Config (</code> <a href="#"><code>I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)</code></a>
Function Description	Configures I2C Analog noise filter.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b> : pointer to a <code>I2C_HandleTypeDef</code> structure that contains the configuration information for the specified I2Cx peripheral.</li> <li>• <b>AnalogFilter</b> : new state of the Analog filter.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 24.1.4.2 HAL\_I2CEEx\_DigitalFilter\_Config

Function Name	<b>HAL_StatusTypeDef HAL_I2CEEx_DigitalFilter_Config ( <i>I2C_HandleTypeDef</i> * hi2c, uint32_t DigitalFilter)</b>
Function Description	Configures I2C Digital noise filter.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2c</b> : pointer to a <i>I2C_HandleTypeDef</i> structure that contains the configuration information for the specified I2Cx peripheral.</li><li>• <b>DigitalFilter</b> : Coefficient of digital noise filter between 0x00 and 0x0F.</li></ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 24.2 I2CEEx Firmware driver defines

### 24.2.1 I2CEEx

I2CEEx

#### *I2CEEx\_Analog\_Filter*

- #define: ***I2C\_ANALOGFILTER\_ENABLED* ((uint32\_t)0x00000000)**
- #define: ***I2C\_ANALOGFILTER\_DISABLED* *I2C\_FLTR\_ANOFF***

## 25 HAL I2S Generic Driver

### 25.1 I2S Firmware driver registers structures

#### 25.1.1 I2S\_HandleTypeDef

*I2S\_HandleTypeDef* is defined in the `stm32f4xx_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_InitTypeDef I2S_HandleTypeDef::Init`
- `uint16_t* I2S_HandleTypeDef::pTxBuffPtr`
- `__IO uint16_t I2S_HandleTypeDef::TxXferSize`
- `__IO uint16_t I2S_HandleTypeDef::TxXferCount`
- `uint16_t* I2S_HandleTypeDef::pRxBuffPtr`
- `__IO uint16_t I2S_HandleTypeDef::RxXferSize`
- `__IO uint16_t I2S_HandleTypeDef::RxXferCount`
- `DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx`
- `__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock`
- `__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State`
- `__IO HAL_I2S_ErrorTypeDef I2S_HandleTypeDef::ErrorCode`

#### 25.1.2 I2S\_InitTypeDef

*I2S\_InitTypeDef* is defined in the `stm32f4xx_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`

### 25.1.3 SPI\_TypeDef

`SPI_TypeDef` is defined in the `stm32f439xx.h`

#### Data Fields

- `__IO uint32_t CR1`
- `__IO uint32_t CR2`
- `__IO uint32_t SR`
- `__IO uint32_t DR`
- `__IO uint32_t CRCPR`
- `__IO uint32_t RXCRCR`
- `__IO uint32_t TXCRCR`
- `__IO uint32_t I2SCFGR`
- `__IO uint32_t I2SPR`

## Field Documentation

- `__IO uint32_t SPI_TypeDef::CR1`
  - SPI control register 1 (not used in I2S mode), Address offset: 0x00
- `__IO uint32_t SPI_TypeDef::CR2`
  - SPI control register 2, Address offset: 0x04
- `__IO uint32_t SPI_TypeDef::SR`
  - SPI status register, Address offset: 0x08
- `__IO uint32_t SPI_TypeDef::DR`
  - SPI data register, Address offset: 0x0C
- `__IO uint32_t SPI_TypeDef::CRCPR`
  - SPI CRC polynomial register (not used in I2S mode), Address offset: 0x10
- `__IO uint32_t SPI_TypeDef::RXCRCR`
  - SPI RX CRC register (not used in I2S mode), Address offset: 0x14
- `__IO uint32_t SPI_TypeDef::TXCRCR`
  - SPI TX CRC register (not used in I2S mode), Address offset: 0x18
- `__IO uint32_t SPI_TypeDef::I2SCFGR`
  - SPI\_I2S configuration register, Address offset: 0x1C
- `__IO uint32_t SPI_TypeDef::I2SPR`
  - SPI\_I2S prescaler register, Address offset: 0x20

## 25.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

### 25.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S\_HandleTypeDef handle structure.
2. Initialize the I2S low level resources by implement the HAL\_I2S\_MspInit() API:
  - a. Enable the SPIx interface clock.
  - b. I2S pins configuration:
    - Enable the clock for the I2S GPIOs.
    - Configure these I2S pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (HAL\_I2S\_Transmit\_IT() and HAL\_I2S\_Receive\_IT() APIs).
    - Configure the I2Sx interrupt priority.
    - Enable the NVIC I2S IRQ handle.
  - d. DMA Configuration if you need to use DMA process (HAL\_I2S\_Transmit\_DMA() and HAL\_I2S\_Receive\_DMA() APIs):
    - Declare a DMA handle structure for the Tx/Rx stream.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.
    - Configure the DMA Tx/Rx Stream.
    - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.

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 \_\_I2S\_ENABLE\_IT() and \_\_I2S\_DISABLE\_IT() inside the transmit and receive process. Make sure that either: I2S PLL is configured or External clock source is configured after setting correctly the define constant EXTERNAL\_CLOCK\_VALUE in the stm32f4xx\_hal\_conf.h file.
4. Three operation modes are available within this driver :

### Polling mode IO operation

- Send an amount of data in blocking mode using HAL\_I2S\_Transmit()
- Receive an amount of data in blocking mode using HAL\_I2S\_Receive()

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL\_I2S\_Transmit\_IT()
- At transmission end of half transfer HAL\_I2S\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxHalfCpltCallback
- At transmission end of transfer HAL\_I2S\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_I2S\_Receive\_IT()
- At reception end of half transfer HAL\_I2S\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxHalfCpltCallback
- At reception end of transfer HAL\_I2S\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxCpltCallback
- In case of transfer Error, HAL\_I2S\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2S\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_I2S\_Transmit\_DMA()
- At transmission end of half transfer HAL\_I2S\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxHalfCpltCallback
- At transmission end of transfer HAL\_I2S\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_I2S\_Receive\_DMA()
- At reception end of half transfer HAL\_I2S\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxHalfCpltCallback
- At reception end of transfer HAL\_I2S\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2S\_RxCpltCallback
- In case of transfer Error, HAL\_I2S\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2S\_ErrorCallback
- Pause the DMA Transfer using HAL\_I2S\_DMAPause()
- Resume the DMA Transfer using HAL\_I2S\_DMAResume()

- Stop the DMA Transfer using HAL\_I2S\_DMAStop()

### I2S HAL driver macros list

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

- `_HAL_I2S_ENABLE`: Enable the specified SPI peripheral (in I2S mode)
- `_HAL_I2S_DISABLE`: Disable the specified SPI peripheral (in I2S mode)
- `_HAL_I2S_ENABLE_IT` : Enable the specified I2S interrupts
- `_HAL_I2S_DISABLE_IT` : Disable the specified I2S interrupts
- `_HAL_I2S_GET_FLAG`: Check whether the specified I2S flag is set or not



You can refer to the I2S HAL driver header file for more useful macros

### 25.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize 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\_DelInit() to restore the default configuration of the selected I2Sx peripheral.
- **`HAL_I2S_Init()`**
- **`HAL_I2S_DelInit()`**
- **`HAL_I2S_MspInit()`**
- **`HAL_I2S_MspDelInit()`**

### 25.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :

- HAL\_I2S\_Transmit()
- HAL\_I2S\_Receive()
- 3. No-Blocking mode functions with Interrupt are :
  - HAL\_I2S\_Transmit\_IT()
  - HAL\_I2S\_Receive\_IT()
- 4. No-Blocking mode functions with DMA are :
  - HAL\_I2S\_Transmit\_DMA()
  - HAL\_I2S\_Receive\_DMA()
- 5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_I2S\_TxCpltCallback()
  - HAL\_I2S\_RxCpltCallback()
  - HAL\_I2S\_ErrorCallback()
  - *HAL\_I2S\_Transmit()*
  - *HAL\_I2S\_Receive()*
  - *HAL\_I2S\_Transmit\_IT()*
  - *HAL\_I2S\_Receive\_IT()*
  - *HAL\_I2S\_Transmit\_DMA()*
  - *HAL\_I2S\_Receive\_DMA()*
  - *HAL\_I2S\_DMAPause()*
  - *HAL\_I2S\_DMAResume()*
  - *HAL\_I2S\_DMAPause()*
  - *HAL\_I2S\_DMAResume()*
  - *HAL\_I2S\_IRQHandler()*
  - *HAL\_I2S\_TxHalfCpltCallback()*
  - *HAL\_I2S\_TxCpltCallback()*
  - *HAL\_I2S\_RxHalfCpltCallback()*
  - *HAL\_I2S\_RxCpltCallback()*
  - *HAL\_I2S\_ErrorCallback()*

## 25.2.4 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()*

## 25.2.5 Initialization and de-initialization functions

### 25.2.5.1 HAL\_I2S\_Init

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Init ( <i>I2S_HandleTypeDef</i> * <b>hi2s</b>)</b>
Function Description	Initializes the I2S according to the specified parameters in the <i>I2S_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 25.2.5.2 HAL\_I2S\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DelInit ( <i>I2S_HandleTypeDef</i> * hi2s)</b>
Function Description	Deinitializes the I2S peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 25.2.5.3 HAL\_I2S\_MspInit

Function Name	<b>void HAL_I2S_MspInit ( <i>I2S_HandleTypeDef</i> * hi2s)</b>
Function Description	I2S MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 25.2.5.4 HAL\_I2S\_MspDelInit

Function Name	<b>void HAL_I2S_MspDelInit ( <i>I2S_HandleTypeDef</i> * hi2s)</b>
Function Description	I2S MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a <i>I2S_HandleTypeDef</i> structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 25.2.6 IO operation functions

### 25.2.6.1 HAL\_I2S\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit ( <i>I2S_HandleTypeDef</i> * * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData</b> : a 16-bit pointer to data buffer.</li> <li>• <b>Size</b> : number of data sample to be sent:</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the <b>Size</b> 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 <b>Size</b> parameter means the number of 16-bit data length.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

### 25.2.6.2 HAL\_I2S\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive ( <i>I2S_HandleTypeDef</i> * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData</b> : a 16-bit pointer to data buffer.</li> <li>• <b>Size</b> : number of data sample to be sent:</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the <b>Size</b></li> </ul>

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 continuous way and as the I2S is not disabled at the end of the I2S transaction.

### 25.2.6.3 HAL\_I2S\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_IT (</b> <b>I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
Function Description	Transmit an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData</b> : a 16-bit pointer to data buffer.</li> <li>• <b>Size</b> : number of data sample to be sent:</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

### 25.2.6.4 HAL\_I2S\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_IT (</b> <b>I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"> <li>• <b>pData</b> : a 16-bit pointer to the Receive data buffer.</li> <li>• <b>Size</b> : number of data sample to be sent:</li> </ul>  |
| Notes         | <ul style="list-style-type: none"> <li>• <b>HAL status</b></li> <li>• 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.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> <li>• 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.</li> </ul> |

### 25.2.6.5 HAL\_I2S\_Transmit\_DMA

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_I2S_Transmit_DMA (</b><br><b>I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>  |
| Function Description | Transmit an amount of data in non-blocking mode with DMA.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData</b> : a 16-bit pointer to the Transmit data buffer.</li> <li>• <b>Size</b> : number of data sample to be sent:</li> </ul>  |
| Return values        | <ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>   |
| Notes                | <ul style="list-style-type: none"> <li>• 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.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul> |

### 25.2.6.6 HAL\_I2S\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_DMA (</b> <b>I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pData</b> : a 16-bit pointer to the Receive data buffer.</li> <li>• <b>Size</b> : number of data sample to be sent:</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

### 25.2.6.7 HAL\_I2S\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAPause (</b> <b>I2S_HandleTypeDef * hi2s)</b>
Function Description	Pauses the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 25.2.6.8 HAL\_I2S\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAResume (</b> <b>I2S_HandleTypeDef * hi2s)</b>
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul> |
| Notes         | <ul style="list-style-type: none"> <li>• None.</li> </ul>             |

### 25.2.6.9 HAL\_I2S\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAStop ( <i>I2S_HandleTypeDef</i> * <i>hi2s</i>)</b>
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 25.2.6.10 HAL\_I2S\_IRQHandler

Function Name	<b>void HAL_I2S_IRQHandler ( <i>I2S_HandleTypeDef</i> * <i>hi2s</i>)</b>
Function Description	This function handles I2S interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 25.2.6.11 HAL\_I2S\_TxHalfCpltCallback

Function Name	<b>void HAL_I2S_TxHalfCpltCallback ( <i>I2S_HandleTypeDef</i> * <i>hi2s</i>)</b>
Function Description	Tx Transfer Half completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that</li> </ul>

contains the configuration information for I2S module

Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 25.2.6.12 HAL\_I2S\_TxCpltCallback

Function Name            **void HAL\_I2S\_TxCpltCallback ( *I2S\_HandleTypeDef* \* hi2s)**

Function Description    Tx Transfer completed callbacks.

Parameters	<ul style="list-style-type: none"><li><b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li></ul>
------------	---

Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 25.2.6.13 HAL\_I2S\_RxHalfCpltCallback

Function Name            **void HAL\_I2S\_RxHalfCpltCallback ( *I2S\_HandleTypeDef* \* hi2s)**

Function Description    Rx Transfer half completed callbacks.

Parameters	<ul style="list-style-type: none"><li><b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li></ul>
------------	---

Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 25.2.6.14 HAL\_I2S\_RxCpltCallback

Function Name            **void HAL\_I2S\_RxCpltCallback ( *I2S\_HandleTypeDef* \* hi2s)**

Function Description    Rx Transfer completed callbacks.

Parameters	<ul style="list-style-type: none"><li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 25.2.6.15 HAL\_I2S\_ErrorCallback

Function Name	<b>void HAL_I2S_ErrorCallback ( <i>I2S_HandleTypeDef</i> * hi2s)</b>
Function Description	I2S error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 25.2.7 Peripheral State and Errors functions

### 25.2.7.1 HAL\_I2S\_GetState

Function Name	<b>HAL_I2S_StateTypeDef HAL_I2S_GetState ( <i>I2S_HandleTypeDef</i> * hi2s)</b>
Function Description	Return the I2S state.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 25.2.7.2 HAL\_I2S\_GetError

Function Name	<b>HAL_I2S_ErrorTypeDef HAL_I2S_GetError (</b> <b>I2S_HandleTypeDef * hi2s)</b>
Function Description	Return the I2S error code.
Parameters	<ul style="list-style-type: none"><li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>I2S Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 25.3 I2S Firmware driver defines

### 25.3.1 I2S

I2S

*I2S\_Audio\_Frequency*

- #define: ***I2S\_AUDIOFREQ\_192K ((uint32\_t)192000)***
- #define: ***I2S\_AUDIOFREQ\_96K ((uint32\_t)96000)***
- #define: ***I2S\_AUDIOFREQ\_48K ((uint32\_t)48000)***
- #define: ***I2S\_AUDIOFREQ\_44K ((uint32\_t)44100)***
- #define: ***I2S\_AUDIOFREQ\_32K ((uint32\_t)32000)***
- #define: ***I2S\_AUDIOFREQ\_22K ((uint32\_t)22050)***
- #define: ***I2S\_AUDIOFREQ\_16K ((uint32\_t)16000)***
- #define: ***I2S\_AUDIOFREQ\_11K ((uint32\_t)11025)***

- #define: *I2S\_AUDIOFREQ\_8K* ((*uint32\_t*)8000)

- #define: *I2S\_AUDIOFREQ\_DEFAULT* ((*uint32\_t*)2)

#### *I2S\_Clock\_Polarity*

- #define: *I2S\_CPOL\_LOW* ((*uint32\_t*)0x00000000)
- #define: *I2S\_CPOL\_HIGH* ((*uint32\_t*)*SPI\_I2SCFGR\_CKPOL*)

#### *I2S\_Clock\_Source*

- #define: *I2S\_CLOCK\_PLL* ((*uint32\_t*)0x00000000)
- #define: *I2S\_CLOCK\_EXTERNAL* ((*uint32\_t*)0x00000001)

#### *I2S\_Data\_Format*

- #define: *I2S\_DATAFORMAT\_16B* ((*uint32\_t*)0x00000000)
- #define: *I2S\_DATAFORMAT\_16B\_EXTENDED* ((*uint32\_t*)0x00000001)
- #define: *I2S\_DATAFORMAT\_24B* ((*uint32\_t*)0x00000003)
- #define: *I2S\_DATAFORMAT\_32B* ((*uint32\_t*)0x00000005)

#### *I2S\_Flag\_definition*

- #define: **I2S\_FLAG\_TXE SPI\_SR\_TXE**
- #define: **I2S\_FLAG\_RXNE SPI\_SR\_RXNE**
- #define: **I2S\_FLAG\_UDR SPI\_SR\_UDR**
- #define: **I2S\_FLAG\_OVR SPI\_SR\_OVR**
- #define: **I2S\_FLAG\_FRE SPI\_SR\_FRE**
- #define: **I2S\_FLAG\_CHSIDE SPI\_SR\_CHSIDE**
- #define: **I2S\_FLAG\_BSY SPI\_SR\_BSY**

#### ***I2S\_FullDuplex\_Mode***

- #define: **I2S\_FULLDUPLEXMODE\_DISABLE ((uint32\_t)0x00000000)**
- #define: **I2S\_FULLDUPLEXMODE\_ENABLE ((uint32\_t)0x00000001)**

#### ***I2S Interrupt configuration definition***

- #define: **I2S\_IT\_TXE SPI\_CR2\_TXEIE**
- #define: **I2S\_IT\_RXNE SPI\_CR2\_RXNEIE**
- #define: **I2S\_IT\_ERR SPI\_CR2\_ERRIE**

*I2S\_Legacy*

- #define: *I2S\_STANDARD\_PHILLIPS* *I2S\_STANDARD\_PHILIPS*

*I2S\_MCLK\_Output*

- #define: *I2S\_MCLKOUTPUT\_ENABLE* ((*uint32\_t*)*SPI\_I2SPR\_MCKOE*)
- #define: *I2S\_MCLKOUTPUT\_DISABLE* ((*uint32\_t*)0x00000000)

*I2S\_Mode*

- #define: *I2S\_MODE\_SLAVE\_TX* ((*uint32\_t*)0x00000000)
- #define: *I2S\_MODE\_SLAVE\_RX* ((*uint32\_t*)0x00000100)
- #define: *I2S\_MODE\_MASTER\_TX* ((*uint32\_t*)0x00000200)
- #define: *I2S\_MODE\_MASTER\_RX* ((*uint32\_t*)0x00000300)

*I2S\_Standard*

- #define: *I2S\_STANDARD\_PHILIPS* ((*uint32\_t*)0x00000000)
- #define: *I2S\_STANDARD\_MSB* ((*uint32\_t*)0x00000010)
- #define: *I2S\_STANDARD\_LSB* ((*uint32\_t*)0x00000020)

- #define: *I2S\_STANDARD\_PCM\_SHORT ((uint32\_t)0x00000030)*

- #define: *I2S\_STANDARD\_PCM\_LONG ((uint32\_t)0x000000B0)*

## 26 HAL I2S Extension Driver

### 26.1 I2SEEx Firmware driver API description

The following section lists the various functions of the I2SEEx library.

#### 26.1.1 I2S Extension 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 (i.e I2S2ext for SPI2 and I2S3ext for SPI3).
2. The extension block is not a full SPI IP, it is used only as I2S slave to implement full duplex mode. The extension block uses the same clock sources as its master.
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.

#### 26.1.2 How to use this driver

Three operation modes 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()

### 26.1.3 Extension features 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\_TransmitReceive()
3. No-Blocking mode functions with Interrupt are :
  - HAL\_I2S\_TransmitReceive\_IT()
4. No-Blocking mode functions with DMA are :
  - HAL\_I2S\_TransmitReceive\_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_I2S\_TxCpltCallback()
  - HAL\_I2S\_RxCpltCallback()
  - HAL\_I2S\_ErrorCallback()
  - ***HAL\_I2SEx\_TransmitReceive()***
  - ***HAL\_I2SEx\_TransmitReceive\_IT()***
  - ***HAL\_I2SEx\_TransmitReceive\_DMA()***

### 26.1.4 Extension features functions

#### 26.1.4.1 HAL\_I2SEx\_TransmitReceive

Function Name	<b>HAL_StatusTypeDef HAL_I2SEEx_TransmitReceive (</b> <b>I2S_HandleTypeDef * hi2s, uint16_t * pTxData, uint16_t * pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Full-Duplex Transmit/Receive data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li><b>pTxData</b> : a 16-bit pointer to the Transmit data buffer.</li> <li><b>pRxData</b> : a 16-bit pointer to the Receive data buffer.</li> <li><b>Size</b> : number of data sample to be sent:</li> </ul>
Parameters	<ul style="list-style-type: none"> <li><b>Timeout</b> : Timeout duration</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the <b>Size</b> 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 <b>Size</b> parameter means the number of 16-bit data length.</li> <li>The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

#### 26.1.4.2 HAL\_I2SEEx\_TransmitReceive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2SEEx_TransmitReceive_IT (</b> <b>I2S_HandleTypeDef * hi2s, uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)</b>
Function Description	Full-Duplex Transmit/Receive data in non-blocking mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li><b>pTxData</b> : a 16-bit pointer to the Transmit data buffer.</li> <li><b>pRxData</b> : a 16-bit pointer to the Receive data buffer.</li> <li><b>Size</b> : number of data sample to be sent:</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the <b>Size</b> 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 <b>Size</b> parameter means the number of 16-bit data length.</li> <li>The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

audio streaming).

#### 26.1.4.3 HAL\_I2SEEx\_TransmitReceive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2SEEx_TransmitReceive_DMA ( I2S_HandleTypeDef * hi2s, uint16_t * pTxData, uint16_t * pRxData, uint16_t Size)</code>
Function Description	Full-Duplex Transmit/Receive data in non-blocking mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s</b> : pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module</li> <li>• <b>pTxData</b> : a 16-bit pointer to the Transmit data buffer.</li> <li>• <b>pRxData</b> : a 16-bit pointer to the Receive data buffer.</li> <li>• <b>Size</b> : number of data sample to be sent:</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the <b>Size</b> 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 <b>Size</b> parameter means the number of 16-bit data length.</li> <li>• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).</li> </ul>

## 26.2 I2SEEx Firmware driver defines

### 26.2.1 I2SEEx

I2SEEx

## 27 HAL IRDA Generic Driver

### 27.1 IRDA Firmware driver registers structures

#### 27.1.1 IRDA\_HandleTypeDef

*IRDA\_HandleTypeDef* is defined in the `stm32f4xx_hal_irda.h`

##### Data Fields

- `USART_TypeDef * Instance`
- `IRDA_InitTypeDef Init`
- `uint8_t * pTxBuffPtr`
- `uint16_t TxXferSize`
- `uint16_t TxXferCount`
- `uint8_t * pRxBuffPtr`
- `uint16_t RxXferSize`
- `uint16_t RxXferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `__IO HAL_IRDA_StateTypeDef State`
- `__IO HAL_IRDA_ErrorTypeDef ErrorCode`

##### Field Documentation

- `USART_TypeDef* IRDA_HandleTypeDef::Instance`
- `IRDA_InitTypeDef IRDA_HandleTypeDef::Init`
- `uint8_t* IRDA_HandleTypeDef::pTxBuffPtr`
- `uint16_t IRDA_HandleTypeDef::TxXferSize`
- `uint16_t IRDA_HandleTypeDef::TxXferCount`
- `uint8_t* IRDA_HandleTypeDef::pRxBuffPtr`
- `uint16_t IRDA_HandleTypeDef::RxXferSize`
- `uint16_t IRDA_HandleTypeDef::RxXferCount`
- `DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx`
- `HAL_LockTypeDef IRDA_HandleTypeDef::Lock`
- `__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::State`
- `__IO HAL_IRDA_ErrorTypeDef IRDA_HandleTypeDef::ErrorCode`

#### 27.1.2 IRDA\_InitTypeDef

*IRDA\_InitTypeDef* is defined in the `stm32f4xx_hal_irda.h`

##### Data Fields

- `uint32_t BaudRate`

- *uint32\_t WordLength*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint8\_t Prescaler*
- *uint32\_t IrDAMode*

#### Field Documentation

- *uint32\_t IRDA\_InitTypeDef::BaudRate*
  - This member configures the IRDA communication baud rate. The baud rate is computed using the following formula: IntegerDivider = ((PCLKx) / (8 \* (hirda->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32\_t) IntegerDivider) \* 8) + 0.5)
- *uint32\_t IRDA\_InitTypeDef::WordLength*
  - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of *IRDA\_Word\_Length*
- *uint32\_t IRDA\_InitTypeDef::Parity*
  - Specifies the parity mode. This parameter can be a value of *IRDA\_Parity*
- *uint32\_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
- *uint32\_t IRDA\_InitTypeDef::IrDAMode*
  - Specifies the IrDA mode. This parameter can be a value of *IrDA\_Low\_Power*

### 27.1.3 USART\_TypeDef

*USART\_TypeDef* is defined in the stm32f439xx.h

#### Data Fields

- *\_\_IO uint32\_t SR*
- *\_\_IO uint32\_t DR*
- *\_\_IO uint32\_t BRR*
- *\_\_IO uint32\_t CR1*
- *\_\_IO uint32\_t CR2*
- *\_\_IO uint32\_t CR3*
- *\_\_IO uint32\_t GTPR*

#### Field Documentation

- *\_\_IO uint32\_t USART\_TypeDef::SR*
  - USART Status register, Address offset: 0x00
- *\_\_IO uint32\_t USART\_TypeDef::DR*
  - USART Data register, Address offset: 0x04
- *\_\_IO uint32\_t USART\_TypeDef::BRR*
  - USART Baud rate register, Address offset: 0x08

- `__IO uint32_t USART_TypeDef::CR1`
  - USART Control register 1, Address offset: 0x0C
- `__IO uint32_t USART_TypeDef::CR2`
  - USART Control register 2, Address offset: 0x10
- `__IO uint32_t USART_TypeDef::CR3`
  - USART Control register 3, Address offset: 0x14
- `__IO uint32_t USART_TypeDef::GTPR`
  - USART Guard time and prescaler register, Address offset: 0x18

## 27.2 IRDA Firmware driver API description

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

### 27.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA\_HandleTypeDef handle structure.
2. Initialize the IRDA low level resources by implementing the HAL\_IRDA\_MspInit() API:
  - a. Enable the USARTx interface clock.
  - b. IRDA pins configuration:
    - Enable the clock for the IRDA GPIOs.
    - Configure these IRDA 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 interrupt priority.
    - Enable the NVIC USART IRQ handle.
  - d. DMA Configuration if you need to use DMA process (HAL\_IRDA\_Transmit\_DMA() and HAL\_IRDA\_Receive\_DMA() APIs):
    - Declare a DMA handle structure for the Tx/Rx stream.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.
    - Configure the DMA Tx/Rx Stream.
    - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. Program the Baud Rate, Word Length, Parity, IrDA Mode, Prescaler and Mode(Receiver/Transmitter) in the hirda Init structure.
4. Initialize the IRDA registers by calling the HAL\_IRDA\_Init() API:
  - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc by calling the customized 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



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

## 27.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in IrDA mode.

- For the asynchronous mode only these parameters can be configured:
  - BaudRate
  - WordLength
  - 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. Refer to STM32F4xx reference manual (RM0090) for the IRDA frame formats depending on the frame length defined by the M bit (8-bits or 9-bits).
  - Prescaler: A pulse of width less than two and greater than one PSC period(s) may or may not be rejected. The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).
  - Mode: Receiver/transmitter modes
  - IrDAMode: the IrDA can operate in the Normal mode or in the Low power mode.

The HAL\_IRDA\_Init() API 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\(\)\*](#)

### 27.2.3 IO operation functions

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA. These APIs return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_IRDA\_TxCpltCallback(), HAL\_IRDA\_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process. The HAL\_IRDA\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode API's are :
  - [\*HAL\\_IRDA\\_Transmit\(\)\*](#)
  - [\*HAL\\_IRDA\\_Receive\(\)\*](#)
3. Non Blocking mode APIs with Interrupt are :
  - [\*HAL\\_IRDA\\_Transmit\\_IT\(\)\*](#)
  - [\*HAL\\_IRDA\\_Receive\\_IT\(\)\*](#)
  - [\*HAL\\_IRDA\\_IRQHandler\(\)\*](#)
4. Non Blocking mode functions with DMA are :
  - [\*HAL\\_IRDA\\_Transmit\\_DMA\(\)\*](#)
  - [\*HAL\\_IRDA\\_Receive\\_DMA\(\)\*](#)
5. A set of Transfer Complete Callbacks are provided in non 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\(\)\*](#)

### 27.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- HAL\_IRDA\_GetState() API can be helpful to check in run-time the state of the IrDA peripheral.
- HAL\_IRDA\_GetError() check in run-time errors that could be occurred during communication.
- ***HAL\_IRDA\_GetState()***
- ***HAL\_IRDA\_GetError()***

## 27.2.5 IrDA Initialization and de-initialization functions

### 27.2.5.1 HAL\_IRDA\_Init

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Init ( <i>IRDA_HandleTypeDef</i> * <i>hirda</i>)</b>
Function Description	Initializes the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b><i>hirda</i></b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 27.2.5.2 HAL\_IRDA\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DeInit ( <i>IRDA_HandleTypeDef</i> * <i>hirda</i>)</b>
Function Description	DeInitializes the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b><i>hirda</i></b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 27.2.5.3 HAL\_IRDA\_MspInit

Function Name	<b>void HAL_IRDA_MspInit ( <i>IRDA_HandleTypeDef</i> * hirda)</b>
Function Description	IRDA MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 27.2.5.4 HAL\_IRDA\_MspDelInit

Function Name	<b>void HAL_IRDA_MspDelInit ( <i>IRDA_HandleTypeDef</i> * hirda)</b>
Function Description	IRDA MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 27.2.6 IO operation functions

### 27.2.6.1 HAL\_IRDA\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit ( <i>IRDA_HandleTypeDef</i> * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Sends an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>

- **Timeout** : Specify timeout value
  - **HAL status**
  - None.
- Return values
- Notes

### 27.2.6.2 HAL\_IRDA\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive (</b> <b><i>IRDA_HandleTypeDef</i> * hirda, uint8_t * pData, uint16_t Size,</b> <b><i>uint32_t</i> Timeout)</b>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> <li>• <b>Timeout</b> : Specify timeout value</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 27.2.6.3 HAL\_IRDA\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_IT (</b> <b><i>IRDA_HandleTypeDef</i> * hirda, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 27.2.6.4 HAL\_IRDA\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_IT (</b> <i>IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</i>
Function Description	Receives an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 27.2.6.5 HAL\_IRDA\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (</b> <i>IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</i>
Function Description	Sends an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 27.2.6.6 HAL\_IRDA\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_DMA (</b> <i>IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)</i>
Function Description	Receives an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA</li> </ul>

---

	module.
• <b>pData</b> : Pointer to data buffer	
• <b>Size</b> : Amount of data to be received	
Return values	• <b>HAL status</b>
Notes	• When the IRDA parity is enabled (PCE = 1) the data received contain the parity bit.

### 27.2.6.7 HAL\_IRDA\_IRQHandler

Function Name	<b>void HAL_IRDA_IRQHandler ( <i>IRDA_HandleTypeDef</i> * hirda)</b>
Function Description	This function handles IRDA interrupt request.
Parameters	• <b>hirda</b> : pointer to a <i>IRDA_HandleTypeDef</i> structure that contains the configuration information for the specified IRDA module.
Return values	• None.
Notes	• None.

### 27.2.6.8 HAL\_IRDA\_TxCpltCallback

Function Name	<b>void HAL_IRDA_TxCpltCallback ( <i>IRDA_HandleTypeDef</i> * hirda)</b>
Function Description	Tx Transfer complete callbacks.
Parameters	• <b>hirda</b> : pointer to a <i>IRDA_HandleTypeDef</i> structure that contains the configuration information for the specified IRDA module.
Return values	• None.
Notes	• None.

### 27.2.6.9 HAL\_IRDA\_RxCpltCallback

Function Name	<b>void HAL_IRDA_RxCpltCallback ( <i>IRDA_HandleTypeDef</i> * <i>hirda</i>)</b>
Function Description	Rx Transfer complete callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 27.2.6.10 HAL\_IRDA\_ErrorCallback

Function Name	<b>void HAL_IRDA_ErrorCallback ( <i>IRDA_HandleTypeDef</i> * <i>hirda</i>)</b>
Function Description	IRDA error callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 27.2.7 Peripheral State and Errors functions

### 27.2.7.1 HAL\_IRDA\_GetState

Function Name	<b>HAL_IRDA_StateTypeDef HAL_IRDA_GetState ( <i>IRDA_HandleTypeDef</i> * <i>hirda</i>)</b>
Function Description	Returns the IRDA state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 27.2.7.2 HAL\_IRDA\_GetError

Function Name	<code>uint32_t HAL_IRDA_GetError ( <i>IRDA_HandleTypeDef</i> * hirda)</code>
Function Description	Return the IRDA error code.
Parameters	<ul style="list-style-type: none"><li>• <b>hirda</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>IRDA Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 27.3 IRDA Firmware driver defines

### 27.3.1 IRDA

IRDA

#### *IRDA\_Flags*

- #define: `IRDA_FLAG_TXE ((uint32_t)0x00000080)`
- #define: `IRDA_FLAG_TC ((uint32_t)0x00000040)`
- #define: `IRDA_FLAG_RXNE ((uint32_t)0x00000020)`
- #define: `IRDA_FLAG_IDLE ((uint32_t)0x00000010)`
- #define: `IRDA_FLAG_ORE ((uint32_t)0x00000008)`
- #define: `IRDA_FLAG_NE ((uint32_t)0x00000004)`

- #define: ***IRDA\_FLAG\_FE*** ((*uint32\_t*)0x00000002)
- #define: ***IRDA\_FLAG\_PE*** ((*uint32\_t*)0x00000001)

#### *IRDA Interrupt definition*

- #define: ***IRDA\_IT\_PE*** ((*uint32\_t*)0x10000100)
- #define: ***IRDA\_IT\_TXE*** ((*uint32\_t*)0x10000080)
- #define: ***IRDA\_IT\_TC*** ((*uint32\_t*)0x10000040)
- #define: ***IRDA\_IT\_RXNE*** ((*uint32\_t*)0x10000020)
- #define: ***IRDA\_IT\_IDLE*** ((*uint32\_t*)0x10000010)
- #define: ***IRDA\_IT\_LBD*** ((*uint32\_t*)0x20000040)
- #define: ***IRDA\_IT\_CTS*** ((*uint32\_t*)0x30000400)
- #define: ***IRDA\_IT\_ERR*** ((*uint32\_t*)0x30000001)

#### *IrDA\_Low\_Power*

- #define: ***IRDA\_POWERMODE\_LOWPOWER*** ((*uint32\_t*)USART\_CR3\_IRLP)

- #define: ***IRDA\_POWERMODE\_NORMAL ((uint32\_t)0x00000000)***

#### ***IRDA\_Mode***

- #define: ***IRDA\_MODE\_RX ((uint32\_t)USART\_CR1\_RE)***
- #define: ***IRDA\_MODE\_TX ((uint32\_t)USART\_CR1\_TE)***
- #define: ***IRDA\_MODE\_TX\_RX ((uint32\_t)(USART\_CR1\_TE | USART\_CR1\_RE))***

#### ***IRDA\_Parity***

- #define: ***IRDA\_PARITY\_NONE ((uint32\_t)0x00000000)***
- #define: ***IRDA\_PARITY\_EVEN ((uint32\_t)USART\_CR1\_PCE)***
- #define: ***IRDA\_PARITY\_ODD ((uint32\_t)(USART\_CR1\_PCE | USART\_CR1\_PS))***

#### ***IRDA\_Word\_Length***

- #define: ***IRDA\_WORDLENGTH\_8B ((uint32\_t)0x00000000)***
- #define: ***IRDA\_WORDLENGTH\_9B ((uint32\_t)USART\_CR1\_M)***

## 28 HAL IWDG Generic Driver

### 28.1 IWDG Firmware driver registers structures

#### 28.1.1 IWDG\_HandleTypeDef

*IWDG\_HandleTypeDef* is defined in the `stm32f4xx_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

#### 28.1.2 IWDG\_InitTypeDef

*IWDG\_InitTypeDef* is defined in the `stm32f4xx_hal_iwdg.h`

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Reload*

##### Field Documentation

- *uint32\_t IWDG\_InitTypeDef::Prescaler*
  - Select the prescaler of the IWDG. This parameter can be a value of *IWDG\_Prescaler*
- *uint32\_t IWDG\_InitTypeDef::Reload*
  - Specifies the IWDG down-counter reload value. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0xFFFF

### 28.1.3 IWDG\_TypeDef

*IWDG\_TypeDef* is defined in the stm32f439xx.h

#### Data Fields

- `__IO uint32_t KR`
- `__IO uint32_t PR`
- `__IO uint32_t RLR`
- `__IO uint32_t SR`

#### Field Documentation

- `__IO uint32_t IWDG_TypeDef::KR`
  - IWDG Key register, Address offset: 0x00
- `__IO uint32_t IWDG_TypeDef::PR`
  - IWDG Prescaler register, Address offset: 0x04
- `__IO uint32_t IWDG_TypeDef::RLR`
  - IWDG Reload register, Address offset: 0x08
- `__IO uint32_t IWDG_TypeDef::SR`
  - IWDG Status register, Address offset: 0x0C

## 28.2 IWDG Firmware driver API description

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

### 28.2.1 IWDG Generic 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 0xFFFF. 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 @32KHz (LSI): ~125us / ~32.7s The IWDG timeout may vary due to LSI frequency dispersion. STM32F4xx devices provide the capability to measure the LSI frequency (LSI clock connected internally to TIM5 CH4 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

## 28.2.2 How to use this driver

- Use IWDG using HAL\_IWDG\_Start() function to:
  - Enable write access to IWDG\_PR and IWDG\_RLR registers.
  - Configure the IWDG prescaler and counter reload values.
  - Reload IWDG counter with value defined in the IWDG\_RLR register.
  - Start the IWDG, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).
- Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_IWDG\_Refresh() function.

### IWDG HAL driver macros list

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

- `_HAL_IWDG_START`: Enable the IWDG peripheral
- `_HAL_IWDG_RELOAD_COUNTER`: Reloads IWDG counter with value defined in the reload register
- `_HAL_IWDG_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
- `_HAL_IWDG_CLEAR_FLAG`: Clear the IWDG's pending flags

## 28.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG\_InitTypeDef and create the associated handle
- Initialize the IWDG MSP
- Deinitialize IWDG MSP
- `HAL_IWDG_Init()`
- `HAL_IWDG_MspInit()`

## 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 Initialization and de-initialization functions

### 28.2.6.1 HAL\_IWDG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Init ( <i>IWDG_HandleTypeDef</i> * <b>hiwdg</b>)</b>
Function Description	Initializes the IWDG according to the specified parameters in the <i>IWDG_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg</b> : pointer to a <i>IWDG_HandleTypeDef</i> structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 28.2.6.2 HAL\_IWDG\_MspInit

Function Name	<b>void HAL_IWDG_MspInit ( <i>IWDG_HandleTypeDef</i> * <b>hiwdg</b>)</b>
Function Description	Initializes the IWDG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hiwdg</b> : pointer to a <i>IWDG_HandleTypeDef</i> structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 28.2.7 IO operation functions

### 28.2.7.1 HAL\_IWDG\_Start

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Start ( <i>IWDG_HandleTypeDef</i> * <b>hiwdg</b>)</b>
Function Description	Starts the IWDG.

Parameters	<ul style="list-style-type: none"><li>• <b>hiwdg</b> : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 28.2.7.2 HAL\_IWDG\_Refresh

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Refresh (</b> <i>IWDG_HandleTypeDef * hiwdg)</i>
Function Description	Refreshes the IWDG.
Parameters	<ul style="list-style-type: none"><li>• <b>hiwdg</b> : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 28.2.8 Peripheral State functions

### 28.2.8.1 HAL\_IWDG\_GetState

Function Name	<b>HAL_IWDG_StateTypeDef HAL_IWDG_GetState (</b> <i>IWDG_HandleTypeDef * hiwdg)</i>
Function Description	Returns the IWDG state.
Parameters	<ul style="list-style-type: none"><li>• <b>hiwdg</b> : pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 28.3 IWDG Firmware driver defines

### 28.3.1 IWDG

IWDG

*IWDG\_Flag\_definition*

- #define: *IWDG\_FLAG\_PVU ((uint32\_t)0x0001)*

*Watchdog counter prescaler value update flag*

- #define: *IWDG\_FLAG\_RVU ((uint32\_t)0x0002)*

*Watchdog counter reload value update flag*

*IWDG\_Prescaler*

- #define: *IWDG\_PRESCALER\_4 ((uint8\_t)0x00)*

*IWDG prescaler set to 4*

- #define: *IWDG\_PRESCALER\_8 ((uint8\_t)0x01)*

*IWDG prescaler set to 8*

- #define: *IWDG\_PRESCALER\_16 ((uint8\_t)0x02)*

*IWDG prescaler set to 16*

- #define: *IWDG\_PRESCALER\_32 ((uint8\_t)0x03)*

*IWDG prescaler set to 32*

- #define: *IWDG\_PRESCALER\_64 ((uint8\_t)0x04)*

*IWDG prescaler set to 64*

- #define: *IWDG\_PRESCALER\_128 ((uint8\_t)0x05)*

*IWDG prescaler set to 128*

- #define: *IWDG\_PRESCALER\_256 ((uint8\_t)0x06)*

*IWDG prescaler set to 256*

*IWDG\_Registers\_BitMask*

- #define: *KR\_KEY\_RELOAD ((uint32\_t)0xAAAA)*

*IWDG reload counter enable*

- #define: **KR\_KEY\_ENABLE** ((*uint32\_t*)0xCCCC)  
*IWDG peripheral enable*
- #define: **KR\_KEY\_EWA** ((*uint32\_t*)0x5555)  
*IWDG KR write Access enable*
- #define: **KR\_KEY\_DWA** ((*uint32\_t*)0x0000)  
*IWDG KR write Access disable*

## 29 HAL LTDC Generic Driver

### 29.1 LTDC Firmware driver registers structures

#### 29.1.1 LTDC\_HandleTypeDef

*LTDC\_HandleTypeDef* is defined in the `stm32f4xx_hal_ltdc.h`

##### Data Fields

- *LTDC\_TypeDef \* Instance*
- *LTDC\_InitTypeDef Init*
- *LTDC\_LayerCfgTypeDef LayerCtg*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_LTDC\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- *LTDC\_TypeDef\* LTDC\_HandleTypeDef::Instance*
  - LTDC Register base address
- *LTDC\_InitTypeDef LTDC\_HandleTypeDef::Init*
  - LTDC parameters
- *LTDC\_LayerCfgTypeDef LTDC\_HandleTypeDef::LayerCtg[MAX\_LAYER]*
  - LTDC Layers parameters
- *HAL\_LockTypeDef LTDC\_HandleTypeDef::Lock*
  - LTDC Lock
- *\_\_IO HAL\_LTDC\_StateTypeDef LTDC\_HandleTypeDef::State*
  - LTDC state
- *\_\_IO uint32\_t LTDC\_HandleTypeDef::ErrorCode*
  - LTDC Error code

#### 29.1.2 LTDC\_InitTypeDef

*LTDC\_InitTypeDef* is defined in the `stm32f4xx_hal_ltdc.h`

##### Data Fields

- *uint32\_t HSPolarity*
- *uint32\_t VSPolarity*
- *uint32\_t DEPolarity*
- *uint32\_t PCPPolarity*
- *uint32\_t HorizontalSync*
- *uint32\_t VerticalSync*
- *uint32\_t AccumulatedHBP*
- *uint32\_t AccumulatedVBP*
- *uint32\_t AccumulatedActiveW*

- *uint32\_t AccumulatedActiveH*
- *uint32\_t TotalWidth*
- *uint32\_t TotalHeigh*
- *LTDC\_ColorTypeDef Backcolor*

### Field Documentation

- *uint32\_t LTDC\_InitTypeDef::HSPolarity*
  - configures the horizontal synchronization polarity. This parameter can be one value of *LTDC\_HS\_POLARITY*
- *uint32\_t LTDC\_InitTypeDef::VSPolarity*
  - configures the vertical synchronization polarity. This parameter can be one value of *LTDC\_VS\_POLARITY*
- *uint32\_t LTDC\_InitTypeDef::DEPolarity*
  - configures the data enable polarity. This parameter can be one of value of *LTDC\_DE\_POLARITY*
- *uint32\_t LTDC\_InitTypeDef::PCPolarity*
  - configures the pixel clock polarity. This parameter can be one of value of *LTDC\_PC\_POLARITY*
- *uint32\_t LTDC\_InitTypeDef::HorizontalSync*
  - configures the number of Horizontal synchronization width. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF.
- *uint32\_t LTDC\_InitTypeDef::VerticalSync*
  - configures the number of Vertical synchronization heigh. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0x7FF.
- *uint32\_t LTDC\_InitTypeDef::AccumulatedHBP*
  - configures the accumulated horizontal back porch width. This parameter must be a number between Min\_Data = LTDC\_HorizontalSync and Max\_Data = 0xFFFF.
- *uint32\_t LTDC\_InitTypeDef::AccumulatedVBP*
  - configures the accumulated vertical back porch heigh. This parameter must be a number between Min\_Data = LTDC\_VerticalSync and Max\_Data = 0x7FF.
- *uint32\_t LTDC\_InitTypeDef::AccumulatedActiveW*
  - configures the accumulated active width. This parameter must be a number between Min\_Data = LTDC\_AccumulatedHBP and Max\_Data = 0xFFFF.
- *uint32\_t LTDC\_InitTypeDef::AccumulatedActiveH*
  - configures the accumulated active heigh. This parameter must be a number between Min\_Data = LTDC\_AccumulatedVBP and Max\_Data = 0x7FF.
- *uint32\_t LTDC\_InitTypeDef::TotalWidth*
  - configures the total width. This parameter must be a number between Min\_Data = LTDC\_AccumulatedActiveW and Max\_Data = 0xFFFF.
- *uint32\_t LTDC\_InitTypeDef::TotalHeigh*
  - configures the total heigh. This parameter must be a number between Min\_Data = LTDC\_AccumulatedActiveH and Max\_Data = 0x7FF.
- *LTDC\_ColorTypeDef LTDC\_InitTypeDef::Backcolor*
  - Configures the background color.

### 29.1.3 LTDC\_ColorTypeDef

*LTDC\_ColorTypeDef* is defined in the *stm32f4xx\_hal\_ltcd.h*

#### Data Fields

- *uint8\_t Blue*
- *uint8\_t Green*
- *uint8\_t Red*
- *uint8\_t Reserved*

#### Field Documentation

- *uint8\_t LTDC\_ColorTypeDef::Blue*
  - Configures the blue value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- *uint8\_t LTDC\_ColorTypeDef::Green*
  - Configures the green value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- *uint8\_t LTDC\_ColorTypeDef::Red*
  - Configures the red value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- *uint8\_t LTDC\_ColorTypeDef::Reserved*
  - Reserved 0xFF

### 29.1.4 LTDC\_LayerCfgTypeDef

*LTDC\_LayerCfgTypeDef* is defined in the `stm32f4xx_hal_ltdc.h`

#### Data Fields

- *uint32\_t WindowX0*
- *uint32\_t WindowX1*
- *uint32\_t WindowY0*
- *uint32\_t WindowY1*
- *uint32\_t PixelFormat*
- *uint32\_t Alpha*
- *uint32\_t Alpha0*
- *uint32\_t BlendingFactor1*
- *uint32\_t BlendingFactor2*
- *uint32\_t FBStartAdress*
- *uint32\_t ImageWidth*
- *uint32\_t ImageHeight*
- *LTDC\_ColorTypeDef Backcolor*

#### Field Documentation

- *uint32\_t LTDC\_LayerCfgTypeDef::WindowX0*
  - Configures the Window Horizontal Start Position. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF.
- *uint32\_t LTDC\_LayerCfgTypeDef::WindowX1*

- Configures the Window Horizontal Stop Position. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowY0***
  - Configures the Window vertical Start Position. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowY1***
  - Configures the Window vertical Stop Position. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::PixelFormat***
  - Specifies the pixel format. This parameter can be one of value of ***LTDC\_Pixelformat***
- ***uint32\_t LTDC\_LayerCfgTypeDef::Alpha***
  - Specifies the constant alpha used for blending. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::Alpha0***
  - Configures the default alpha value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::BlendingFactor1***
  - Select the blending factor 1. This parameter can be one of value of ***LTDC\_BlendingFactor1***
- ***uint32\_t LTDC\_LayerCfgTypeDef::BlendingFactor2***
  - Select the blending factor 2. This parameter can be one of value of ***LTDC\_BlendingFactor2***
- ***uint32\_t LTDC\_LayerCfgTypeDef::FBStartAdress***
  - Configures the color frame buffer address
- ***uint32\_t LTDC\_LayerCfgTypeDef::ImageWidth***
  - Configures the color frame buffer line length. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0x1FFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::ImageHeight***
  - Specifies the number of line in frame buffer. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0x7FF.
- ***LTDC\_ColorTypeDef LTDC\_LayerCfgTypeDef::Backcolor***
  - Configures the layer background color.

### 29.1.5 **LTDC\_Layer\_TypeDef**

***LTDC\_Layer\_TypeDef*** is defined in the stm32f439xx.h

#### Data Fields

- ***\_\_IO uint32\_t CR***
- ***\_\_IO uint32\_t WHPCR***
- ***\_\_IO uint32\_t WVPCR***
- ***\_\_IO uint32\_t CKCR***
- ***\_\_IO uint32\_t PFCR***
- ***\_\_IO uint32\_t CACR***
- ***\_\_IO uint32\_t DCCR***
- ***\_\_IO uint32\_t BFCR***
- ***uint32\_t RESERVED0***
- ***\_\_IO uint32\_t CFBAR***
- ***\_\_IO uint32\_t CFBLR***
- ***\_\_IO uint32\_t CFBLNR***

- `uint32_t RESERVED1`
- `_IO uint32_t CLUTWR`

#### Field Documentation

- `_IO uint32_t LTDC_Layer_TypeDef::CR`
  - LTDC Layerx Control Register Address offset: 0x84
- `_IO uint32_t LTDC_Layer_TypeDef::WHPCR`
  - LTDC Layerx Window Horizontal Position Configuration Register Address offset: 0x88
- `_IO uint32_t LTDC_Layer_TypeDef::WVPCR`
  - LTDC Layerx Window Vertical Position Configuration Register Address offset: 0x8C
- `_IO uint32_t LTDC_Layer_TypeDef::CKCR`
  - LTDC Layerx Color Keying Configuration Register Address offset: 0x90
- `_IO uint32_t LTDC_Layer_TypeDef::PFCR`
  - LTDC Layerx Pixel Format Configuration Register Address offset: 0x94
- `_IO uint32_t LTDC_Layer_TypeDef::CACR`
  - LTDC Layerx Constant Alpha Configuration Register Address offset: 0x98
- `_IO uint32_t LTDC_Layer_TypeDef::DCCR`
  - LTDC Layerx Default Color Configuration Register Address offset: 0x9C
- `_IO uint32_t LTDC_Layer_TypeDef::BFCR`
  - LTDC Layerx Blending Factors Configuration Register Address offset: 0xA0
- `uint32_t LTDC_Layer_TypeDef::RESERVED0[2]`
  - Reserved
- `_IO uint32_t LTDC_Layer_TypeDef::CFBAR`
  - LTDC Layerx Color Frame Buffer Address Register Address offset: 0xAC
- `_IO uint32_t LTDC_Layer_TypeDef::CFBLR`
  - LTDC Layerx Color Frame Buffer Length Register Address offset: 0xB0
- `_IO uint32_t LTDC_Layer_TypeDef::CFBLNR`
  - LTDC Layerx ColorFrame Buffer Line Number Register Address offset: 0xB4
- `uint32_t LTDC_Layer_TypeDef::RESERVED1[3]`
  - Reserved
- `_IO uint32_t LTDC_Layer_TypeDef::CLUTWR`
  - LTDC Layerx CLUT Write Register Address offset: 0x144

### 29.1.6 LTDC\_TypeDef

`LTDC_TypeDef` is defined in the `stm32f439xx.h`

#### Data Fields

- `uint32_t RESERVED0`
- `_IO uint32_t SSCR`
- `_IO uint32_t BPCR`
- `_IO uint32_t AWCR`
- `_IO uint32_t TWCR`
- `_IO uint32_t GCR`
- `uint32_t RESERVED1`
- `_IO uint32_t SRCR`

- `uint32_t RESERVED2`
- `_IO uint32_t BCCR`
- `uint32_t RESERVED3`
- `_IO uint32_t IER`
- `_IO uint32_t ISR`
- `_IO uint32_t ICR`
- `_IO uint32_t LIPCR`
- `_IO uint32_t CPSR`
- `_IO uint32_t CDSR`

### Field Documentation

- `uint32_t LTDC_TypeDef::RESERVED0[2]`
  - Reserved, 0x00-0x04
- `_IO uint32_t LTDC_TypeDef::SSCR`
  - LTDC Synchronization Size Configuration Register, Address offset: 0x08
- `_IO uint32_t LTDC_TypeDef::BPCR`
  - LTDC Back Porch Configuration Register, Address offset: 0x0C
- `_IO uint32_t LTDC_TypeDef::AWCR`
  - LTDC Active Width Configuration Register, Address offset: 0x10
- `_IO uint32_t LTDC_TypeDef::TWCR`
  - LTDC Total Width Configuration Register, Address offset: 0x14
- `_IO uint32_t LTDC_TypeDef::GCR`
  - LTDC Global Control Register, Address offset: 0x18
- `uint32_t LTDC_TypeDef::RESERVED1[2]`
  - Reserved, 0x1C-0x20
- `_IO uint32_t LTDC_TypeDef::SRCR`
  - LTDC Shadow Reload Configuration Register, Address offset: 0x24
- `uint32_t LTDC_TypeDef::RESERVED2[1]`
  - Reserved, 0x28
- `_IO uint32_t LTDC_TypeDef::BCCR`
  - LTDC Background Color Configuration Register, Address offset: 0x2C
- `uint32_t LTDC_TypeDef::RESERVED3[1]`
  - Reserved, 0x30
- `_IO uint32_t LTDC_TypeDef::IER`
  - LTDC Interrupt Enable Register, Address offset: 0x34
- `_IO uint32_t LTDC_TypeDef::ISR`
  - LTDC Interrupt Status Register, Address offset: 0x38
- `_IO uint32_t LTDC_TypeDef::ICR`
  - LTDC Interrupt Clear Register, Address offset: 0x3C
- `_IO uint32_t LTDC_TypeDef::LIPCR`
  - LTDC Line Interrupt Position Configuration Register, Address offset: 0x40
- `_IO uint32_t LTDC_TypeDef::CPSR`
  - LTDC Current Position Status Register, Address offset: 0x44
- `_IO uint32_t LTDC_TypeDef::CDSR`
  - LTDC Current Display Status Register, Address offset: 0x48

## 29.2 LTDC Firmware driver API description

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

### 29.2.1 How to use this driver

1. Program the required configuration through the following parameters: the LTDC timing, the horizontal and vertical polarity, the pixel clock polarity, Data Enable polarity and the LTDC background color value using HAL\_LTDC\_Init() function
2. Program the required configuration through the following parameters: the pixel format, the blending factors, input alpha value, the window size and the image size using HAL\_LTDC\_ConfigLayer() function for foreground or/and background layer.
3. Optionally, configure and enable the CLUT using HAL\_LTDC\_ConfigCLUT() and HAL\_LTDC\_EnableCLUT functions.
4. Optionally, enable the Dither using HAL\_LTDC\_EnableDither().
5. Optionally, configure and enable the Color keying using HAL\_LTDC\_ConfigColorKeying() and HAL\_LTDC\_EnableColorKeying functions.
6. Optionally, configure LineInterrupt using HAL\_LTDC\_ProgramLineInterrupt() function
7. If needed, reconfigure and change the pixel format value, the alpha value value, the window size, the window position and the layer start address for foreground or/and background layer using respectively the following functions:  
HAL\_LTDC\_SetPixelFormat(), HAL\_LTDC\_SetAlpha(),  
HAL\_LTDC\_SetWindowSize(), HAL\_LTDC\_SetWindowPosition(),  
HAL\_LTDC\_SetAddress.
8. To control LTDC state you can use the following function: HAL\_LTDC\_GetState()

#### LTDC HAL driver macros list

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

- \_\_HAL\_LTDC\_ENABLE: Enable the LTDC.
- \_\_HAL\_LTDC\_DISABLE: Disable the LTDC.
- \_\_HAL\_LTDC\_LAYER\_ENABLE: Enable the LTDC Layer.
- \_\_HAL\_LTDC\_LAYER\_DISABLE: Disable the LTDC Layer.
- \_\_HAL\_LTDC\_RELOAD\_CONFIG: Reload Layer Configuration.
- \_\_HAL\_LTDC\_GET\_FLAG: Get the LTDC pending flags.
- \_\_HAL\_LTDC\_CLEAR\_FLAG: Clear the LTDC pending flags.
- \_\_HAL\_LTDC\_ENABLE\_IT: Enable the specified LTDC interrupts.
- \_\_HAL\_LTDC\_DISABLE\_IT: Disable the specified LTDC interrupts.
- \_\_HAL\_LTDC\_GET\_IT\_SOURCE: Check whether the specified LTDC interrupt has occurred or not.



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

### 29.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC

- De-initialize the LTDC
- `HAL_LTDC_Init()`
- `HAL_LTDC_DelInit()`
- `HAL_LTDC_MspInit()`
- `HAL_LTDC_MspDelInit()`

### 29.2.3 IO operation functions

This section provides function allowing to:

- handle LTDC interrupt request
- `HAL_LTDC_IRQHandler()`
- `HAL_LTDC_ErrorCallback()`
- `HAL_LTDC_LineEvenCallback()`

### 29.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the LTDC foreground or/and background parameters.
- Set the active layer.
- Configure the color keying.
- Configure the C-LUT.
- Enable / Disable the color keying.
- Enable / Disable the C-LUT.
- Update the layer position.
- Update the layer size.
- Update pixel format on the fly.
- Update transparency on the fly.
- Update address on the fly.
- `HAL_LTDC_ConfigLayer()`
- `HAL_LTDC_ConfigColorKeying()`
- `HAL_LTDC_ConfigCLUT()`
- `HAL_LTDC_EnableColorKeying()`
- `HAL_LTDC_DisableColorKeying()`
- `HAL_LTDC_EnableCLUT()`
- `HAL_LTDC_DisableCLUT()`
- `HAL_LTDC_EnableDither()`
- `HAL_LTDC_DisableDither()`
- `HAL_LTDC_SetWindowSize()`
- `HAL_LTDC_SetWindowPosition()`
- `HAL_LTDC_SetPixelFormat()`
- `HAL_LTDC_SetAlpha()`
- `HAL_LTDC_SetAddress()`
- `HAL_LTDC_ProgramLineEvent()`

### 29.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the LTDC state.

- Get error code.
- `HAL_LTDC_GetState()`
- `HAL_LTDC_GetError()`

## 29.2.6 Initialization and Configuration functions

### 29.2.6.1 HAL\_LTDC\_Init

Function Name	<code>HAL_StatusTypeDef HAL_LTDC_Init ( LTDC_HandleTypeDef * hltc)</code>
Function Description	Initializes the LTDC according to the specified parameters in the <code>LTDC_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <code>hltc</code> : pointer to a <code>LTDC_HandleTypeDef</code> structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.6.2 HAL\_LTDC\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_LTDC_DeInit ( LTDC_HandleTypeDef * hltc)</code>
Function Description	Deinitializes the LTDC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <code>hltc</code> : pointer to a <code>LTDC_HandleTypeDef</code> structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.6.3 HAL\_LTDC\_MspInit

Function Name	<code>void HAL_LTDC_MspInit ( LTDC_HandleTypeDef * hltc)</code>
Function Description	Initializes the LTDC MSP.

Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 29.2.6.4 HAL\_LTDC\_MspDeInit

Function Name	<b>void HAL_LTDC_MspDeInit ( LTDC_HandleTypeDef * hltc)</b>
Function Description	DeInitializes the LTDC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 29.2.7 IO operation functions

#### 29.2.7.1 HAL\_LTDC\_IRQHandler

Function Name	<b>void HAL_LTDC_IRQHandler ( LTDC_HandleTypeDef * hltc)</b>
Function Description	Handles LTDC interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 29.2.7.2 HAL\_LTDC\_ErrorCallback

Function Name	<b>void HAL_LTDC_ErrorCallback ( LTDC_HandleTypeDef *</b>
---------------	---

**hltc)**

Function Description	Error LTDC callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**29.2.7.3 HAL\_LTDC\_LineEvenCallback**

Function Name	<b>void HAL_LTDC_LineEvenCallback ( <i>LTDC_HandleTypeDef</i> * hltc)</b>
Function Description	Line Event callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**29.2.8 Peripheral Control functions****29.2.8.1 HAL\_LTDC\_ConfigLayer**

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_ConfigLayer ( <i>LTDC_HandleTypeDef</i> * hltc, <i>LTDC_LayerCfgTypeDef</i> * pLayerCfg, uint32_t LayerIdx)</b>
Function Description	Configure the LTDC Layer according to the specified parameters in the <i>LTDC_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>pLayerCfg</b> : pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.</li> <li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.8.2 HAL\_LTDC\_ConfigColorKeying

Function Name	<code>HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying (</code> <code>    <b>LTDC_HandleTypeDef</b> * hltc, uint32_t RGBValue, uint32_t</code> <code>    LayerIdx)</code>
Function Description	Configure the color keying.
Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a <code>LTDC_HandleTypeDef</code> structure that contains the configuration information for the LTDC.</li><li>• <b>RGBValue</b> : the color key value</li><li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 29.2.8.3 HAL\_LTDC\_ConfigCLUT

Function Name	<code>HAL_StatusTypeDef HAL_LTDC_ConfigCLUT (</code> <code>    <b>LTDC_HandleTypeDef</b> * hltc, uint32_t * pCLUT, uint32_t</code> <code>    CLUTSize, uint32_t LayerIdx)</code>
Function Description	Load the color lookup table.
Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a <code>LTDC_HandleTypeDef</code> structure that contains the configuration information for the LTDC.</li><li>• <b>pCLUT</b> : pointer to the color lookup table address.</li><li>• <b>CLUTSize</b> : the color lookup table size.</li><li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 29.2.8.4 HAL\_LTDC\_EnableColorKeying

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_EnableColorKeying (</b> <i>LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)</i>
Function Description	Enable the color keying.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.8.5 HAL\_LTDC\_DisableColorKeying

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_DisableColorKeying (</b> <i>LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)</i>
Function Description	Disable the color keying.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.8.6 HAL\_LTDC\_EnableCLUT

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_EnableCLUT (</b> <i>LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)</i>
Function Description	Enable the color lookup table.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

### 29.2.8.7 HAL\_LTDC\_DisableCLUT

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_DisableCLUT (</b> <b><i>LTDC_HandleTypeDef</i> * hltc, uint32_t LayerIdx)</b>
Function Description	Disable the color lookup table.
Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li><li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 29.2.8.8 HAL\_LTDC\_EnableDither

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_EnableDither (</b> <b><i>LTDC_HandleTypeDef</i> * hltc)</b>
Function Description	Enables Dither.
Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 29.2.8.9 HAL\_LTDC\_DisableDither

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_DisableDither (</b> <b><i>LTDC_HandleTypeDef</i> * hltc)</b>
Function Description	Disables Dither.

Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.8.10 HAL\_LTDC\_SetWindowSize

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_SetWindowSize (</b> <b>LTDC_HandleTypeDef * hltc, uint32_t XSize, uint32_t YSize,</b> <b>uint32_t LayerIdx)</b>
Function Description	Set the LTDC window size.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>XSize</b> : LTDC Pixel per line</li> <li>• <b>YSize</b> : LTDC Line number</li> <li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.8.11 HAL\_LTDC\_SetWindowPosition

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_SetWindowPosition (</b> <b>LTDC_HandleTypeDef * hltc, uint32_t X0, uint32_t Y0,</b> <b>uint32_t LayerIdx)</b>
Function Description	Set the LTDC window position.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>X0</b> : LTDC window X offset</li> <li>• <b>Y0</b> : LTDC window Y offset</li> <li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.8.12 HAL\_LTDC\_SetPixelFormat

Function Name	<code>HAL_StatusTypeDef HAL_LTDC_SetPixelFormat (     LTDC_HandleTypeDef * hltc, uint32_t Pixelformat, uint32_t     LayerIdx)</code>
Function Description	Reconfigure the pixel format.
Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li><li>• <b>Pixelformat</b> : new pixel format value.</li><li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 29.2.8.13 HAL\_LTDC\_SetAlpha

Function Name	<code>HAL_StatusTypeDef HAL_LTDC_SetAlpha (     LTDC_HandleTypeDef * hltc, uint32_t Alpha, uint32_t     LayerIdx)</code>
Function Description	Reconfigure the layer alpha value.
Parameters	<ul style="list-style-type: none"><li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li><li>• <b>Alpha</b> : new alpha value.</li><li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 29.2.8.14 HAL\_LTDC\_SetAddress

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_SetAddress (</b> <b>LTDC_HandleTypeDef * hltc, uint32_t Address, uint32_t</b> <b>LayerIdx)</b>
Function Description	Reconfigure the frame buffer Address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>Address</b> : new address value.</li> <li>• <b>LayerIdx</b> : LTDC Layer index. This parameter can be one of the following values: 0 or 1.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.8.15 HAL\_LTDC\_ProgramLineEvent

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_ProgramLineEvent (</b> <b>LTDC_HandleTypeDef * hltc, uint32_t Line)</b>
Function Description	Define the position of the line interrupt .
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>Line</b> : Line Interrupt Position.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 29.2.9 Peripheral State and Errors functions

### 29.2.9.1 HAL\_LTDC\_GetState

Function Name	<b>HAL_LTDC_StateTypeDef HAL_LTDC_GetState (</b> <b>LTDC_HandleTypeDef * hltc)</b>
Function Description	Return the LTDC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltc</b> : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 29.2.9.2 HAL\_LTDC\_GetError

Function Name	<code>uint32_t HAL_LTDC_GetError ( <i>LTDC_HandleTypeDef</i> * hltc)</code>
Function Description	Return the LTDC error code.
Parameters	<ul style="list-style-type: none"> <li>• <code>hltc</code> : pointer to a <code>LTDC_HandleTypeDef</code> structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>LTDC Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 29.3 LTDC Firmware driver defines

### 29.3.1 LTDC

LTDC

*LTDC\_Alpha*

- #define: `LTDC_ALPHA LTDC_LxCACR_CONSTA`  
*LTDC Cte Alpha mask*

*LTDC\_BACK\_COLOR*

- #define: `LTDC_COLOR ((uint32_t)0x000000FF)`  
*Color mask*

*LTDC\_BlendingFactor1*

- #define: `LTDC_BLENDING_FACTOR1_CA ((uint32_t)0x00000400)`  
*Blending factor : Cte Alpha*

- #define: `LTDC_BLENDING_FACTOR1_PAxCA ((uint32_t)0x00000600)`  
*Blending factor : Cte Alpha x Pixel Alpha*

*LTDC\_BlendingFactor2*

- #define: `LTDC_BLENDING_FACTOR2_CA ((uint32_t)0x00000005)`

*Blending factor : Cte Alpha*

- #define: **LTDC\_BLENDING\_FACTOR2\_PAxCA** ((uint32\_t)0x00000007)

*Blending factor : Cte Alpha x Pixel Alpha*

#### ***LTDC\_DE\_POLARITY***

- #define: **LTDC\_DEPOLARITY\_AL** ((uint32\_t)0x00000000)

*Data Enable, is active low.*

- #define: **LTDC\_DEPOLARITY\_AH LTDC\_GCR\_DEPOL**

*Data Enable, is active high.*

#### ***LTDC\_Flag***

- #define: **LTDC\_FLAG\_LI LTDC\_ISR\_LIF**

- #define: **LTDC\_FLAG\_FU LTDC\_ISR\_FUIF**

- #define: **LTDC\_FLAG\_TE LTDC\_ISR\_TERRIF**

- #define: **LTDC\_FLAG\_RR LTDC\_ISR\_RRIF**

#### ***LTDC\_HS\_POLARITY***

- #define: **LTDC\_HSPOLARITY\_AL** ((uint32\_t)0x00000000)

*Horizontal Synchronization is active low.*

- #define: **LTDC\_HSPOLARITY\_AH LTDC\_GCR\_HSPOL**

*Horizontal Synchronization is active high.*

#### ***LTDC Interrupts***

- #define: **LTDC\_IT\_LI LTDC\_IER\_LIE**

- #define: **LTDC\_IT\_FU** **LTDC\_IER\_FUIE**
- #define: **LTDC\_IT\_TE** **LTDC\_IER\_TERRIE**
- #define: **LTDC\_IT\_RR** **LTDC\_IER\_RRIE**

#### ***LTDC\_LAYER\_Config***

- #define: **LTDC\_STOPPOSITION** (**LTDC\_LxWHPCR\_WHSPPOS >> 16**)  
*LTDC Layer stop position*
- #define: **LTDC\_STARTPOSITION** **LTDC\_LxWHPCR\_WHSTPOS**  
*LTDC Layer start position*
- #define: **LTDC\_COLOR\_FRAME\_BUFFER** **LTDC\_LxCFBLR\_CFBLL**  
*LTDC Layer Line length*
- #define: **LTDC\_LINE\_NUMBER** **LTDC\_LxCFBLNR\_CFBLNBR**  
*LTDC Layer Line number*

#### ***LTDC\_PC\_POLARITY***

- #define: **LTDC\_PCPOLARITY\_IPC** ((**uint32\_t**)0x00000000)  
*input pixel clock.*
- #define: **LTDC\_PCPOLARITY\_IIPC** **LTDC\_GCR\_PCPOL**  
*inverted input pixel clock.*

#### ***LTDC\_Pixelformat***

- #define: **LTDC\_PIXEL\_FORMAT\_ARGB8888** ((**uint32\_t**)0x00000000)  
*ARGB8888 LTDC pixel format*
- #define: **LTDC\_PIXEL\_FORMAT\_RGB888** ((**uint32\_t**)0x00000001)  
*RGB888 LTDC pixel format*

- #define: **LTDC\_PIXEL\_FORMAT\_RGB565** ((*uint32\_t*)0x00000002)  
*RGB565 LTDC pixel format*
- #define: **LTDC\_PIXEL\_FORMAT\_ARGB1555** ((*uint32\_t*)0x00000003)  
*ARGB1555 LTDC pixel format*
- #define: **LTDC\_PIXEL\_FORMAT\_ARGB4444** ((*uint32\_t*)0x00000004)  
*ARGB4444 LTDC pixel format*
- #define: **LTDC\_PIXEL\_FORMAT\_L8** ((*uint32\_t*)0x00000005)  
*L8 LTDC pixel format*
- #define: **LTDC\_PIXEL\_FORMAT\_AL44** ((*uint32\_t*)0x00000006)  
*AL44 LTDC pixel format*
- #define: **LTDC\_PIXEL\_FORMAT\_AL88** ((*uint32\_t*)0x00000007)  
*AL88 LTDC pixel format*

#### ***LTDC\_SYNC***

- #define: **LTDC\_HORIZONTALSYNC** (*LTDC\_SSCR\_HSW* >> 16)  
*Horizontal synchronization width.*
- #define: **LTDC\_VERTICALSYNC** *LTDC\_SSCR\_VSH*  
*Vertical synchronization height.*

#### ***LTDC\_VS\_POLARITY***

- #define: **LTDC\_VSPOLARITY\_AL** ((*uint32\_t*)0x00000000)  
*Vertical Synchronization is active low.*
- #define: **LTDC\_VSPOLARITY\_AH** *LTDC\_GCR\_VSPOL*  
*Vertical Synchronization is active high.*

## 30 HAL NAND Generic Driver

### 30.1 NAND Firmware driver registers structures

#### 30.1.1 NAND\_HandleTypeDef

*NAND\_HandleTypeDef* is defined in the `stm32f4xx_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

#### 30.1.2 NAND\_AddressTypedef

*NAND\_AddressTypedef* is defined in the `stm32f4xx_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

### 30.1.3 NAND\_IDTypeDef

*NAND\_IDTypeDef* is defined in the `stm32f4xx_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`

## 30.2 NAND Firmware driver API description

The following section lists the various functions of the NAND 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 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.

### 30.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

- `HAL_NAND_Init()`
- `HAL_NAND_DelInit()`
- `HAL_NAND_MspInit()`
- `HAL_NAND_MspDelInit()`
- `HAL_NAND_IRQHandler()`
- `HAL_NAND_ITCallback()`

### 30.2.3 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()`

### 30.2.4 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()`

### 30.2.5 NAND State functions

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

- `HAL_NAND_GetState()`

### 30.2.6 Initialization and de-initialization functions

#### 30.2.6.1 HAL\_NAND\_Init

Function Name	<code>HAL_StatusTypeDef HAL_NAND_Init ( <i>NAND_HandleTypeDef</i> * * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)</code>
Function Description	Perform NAND memory Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a <i>NAND_HandleTypeDef</i> structure that contains the configuration information for NAND module.</li> <li>• <b>ComSpace_Timing</b> : pointer to Common space timing structure</li> <li>• <b>AttSpace_Timing</b> : pointer to Attribute space timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 30.2.6.2 HAL\_NAND\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_NAND_DelInit ( <i>NAND_HandleTypeDef</i> * hnand)</code>
Function Description	Perform NAND memory De-Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a <i>NAND_HandleTypeDef</i> structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 30.2.6.3 HAL\_NAND\_MspInit

Function Name	<code>void HAL_NAND_MspInit ( <i>NAND_HandleTypeDef</i> * hnand)</code>
Function Description	NAND MSP Init.

Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 30.2.6.4 HAL\_NAND\_MspDelInit

Function Name	<b>void HAL_NAND_MspDelInit ( <i>NAND_HandleTypeDef</i> * hnand)</b>
Function Description	NAND MSP DelInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 30.2.6.5 HAL\_NAND\_IRQHandler

Function Name	<b>void HAL_NAND_IRQHandler ( <i>NAND_HandleTypeDef</i> * hnand)</b>
Function Description	This function handles NAND device interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 30.2.6.6 HAL\_NAND\_ITCallback

Function Name	<b>void HAL_NAND_ITCallback ( <i>NAND_HandleTypeDef</i> * hnand)</b>
---------------	--

Function Description	NAND interrupt feature callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 30.2.7 Input and Output functions

#### 30.2.7.1 HAL\_NAND\_Read\_ID

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Read_ID (</b> <b>NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)</b>
Function Description	Read the NAND memory electronic signature.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pNAND_ID</b> : NAND ID structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 30.2.7.2 HAL\_NAND\_Reset

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Reset (</b> <b>NAND_HandleTypeDef * hnand)</b>
Function Description	NAND memory reset.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 30.2.7.3 HAL\_NAND\_Read\_Page

Function Name	<code>HAL_StatusTypeDef HAL_NAND_Read_Page (</code> <code>NAND_HandleTypeDef * hnand, NAND_AddressTypeDef *</code> <code>pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)</code>
Function Description	Read Page(s) from NAND memory block.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pAddress</b> : pointer to NAND address structure</li> <li>• <b>pBuffer</b> : pointer to destination read buffer</li> <li>• <b>NumPageToRead</b> : number of pages to read from block</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 30.2.7.4 HAL\_NAND\_Write\_Page

Function Name	<code>HAL_StatusTypeDef HAL_NAND_Write_Page (</code> <code>NAND_HandleTypeDef * hnand, NAND_AddressTypeDef *</code> <code>pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)</code>
Function Description	Write Page(s) to NAND memory block.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pAddress</b> : pointer to NAND address structure</li> <li>• <b>pBuffer</b> : pointer to source buffer to write</li> <li>• <b>NumPageToWrite</b> : number of pages to write to block</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 30.2.7.5 HAL\_NAND\_Read\_SpareArea

Function Name	<code>HAL_StatusTypeDef HAL_NAND_Read_SpareArea (</code> <code>NAND_HandleTypeDef * hnand, NAND_AddressTypeDef *</code> <code>pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)</code>
---------------	--

Function Description	Read Spare area(s) from NAND memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pAddress</b> : pointer to NAND address structure</li> <li>• <b>pBuffer</b> : pointer to source buffer to write</li> <li>• <b>NumSpareAreaToRead</b> : Number of spare area to read</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 30.2.7.6 HAL\_NAND\_Write\_SpareArea

Function Name	<code>HAL_StatusTypeDef HAL_NAND_Write_SpareArea (</code> <code>NAND_HandleTypeDef * hnand, NAND_AddressTypeDef *</code> <code>pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)</code>
Function Description	Write Spare area(s) to NAND memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pAddress</b> : pointer to NAND address structure</li> <li>• <b>pBuffer</b> : pointer to source buffer to write</li> <li>• <b>NumSpareAreaTowrite</b> : number of spare areas to write to block</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 30.2.7.7 HAL\_NAND\_Erase\_Block

Function Name	<code>HAL_StatusTypeDef HAL_NAND_Erase_Block (</code> <code>NAND_HandleTypeDef * hnand, NAND_AddressTypeDef *</code> <code>pAddress)</code>
Function Description	NAND memory Block erase.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> <li>• <b>pAddress</b> : pointer to NAND address structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

- |       |   |
|-------|---|
| Notes | <ul style="list-style-type: none"><li>• None.</li></ul> |
|-------|---|

### 30.2.7.8 HAL\_NAND\_Read\_Status

Function Name	<code>uint32_t HAL_NAND_Read_Status ( <i>NAND_HandleTypeDef</i> * <b>hnand</b>)</code>
Function Description	NAND memory read status.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b> : pointer to a <i>NAND_HandleTypeDef</i> structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>NAND status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 30.2.7.9 HAL\_NAND\_Address\_Inc

Function Name	<code>uint32_t HAL_NAND_Address_Inc ( <i>NAND_HandleTypeDef</i> * <b>hnand</b>, <i>NAND_AddressTypeDef</i> * <b>pAddress</b>)</code>
Function Description	Increment the NAND memory address.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b> : pointer to a <i>NAND_HandleTypeDef</i> structure that contains the configuration information for NAND module.</li><li>• <b>pAddress</b> : pointer to NAND adress structure</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>The new status of the increment address operation. It can be:</b><ul style="list-style-type: none"><li>– <b>NAND_VALID_ADDRESS</b>: When the new address is valid address</li><li>– <b>NAND_INVALID_ADDRESS</b>: When the new address is invalid address</li></ul></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 30.2.8 Control functions

#### 30.2.8.1 HAL\_NAND\_ECC\_Enable

Function Name	<b>HAL_StatusTypeDef HAL_NAND_ECC_Enable (</b> <b>NAND_HandleTypeDef * hnand)</b>
Function Description	Enables dynamically NAND ECC feature.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 30.2.8.2 HAL\_NAND\_ECC\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_NAND_ECC_Disable (</b> <b>NAND_HandleTypeDef * hnand)</b>
Function Description	Disables dynamically FMC_NAND ECC feature.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 30.2.8.3 HAL\_NAND\_GetECC

Function Name	<b>HAL_StatusTypeDef HAL_NAND_GetECC (</b> <b>NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)</b>
Function Description	Disables dynamically NAND ECC feature.
Parameters	<ul style="list-style-type: none"><li>• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li><li>• <b>ECCval</b> : pointer to ECC value</li><li>• <b>Timeout</b> : maximum timeout to wait</li></ul>

---

Return values	• <b>HAL status</b>
Notes	• None.

### 30.2.9 State functions

#### 30.2.9.1 HAL\_NAND\_GetState

Function Name	<b>HAL_NAND_StateTypeDef HAL_NAND_GetState (</b> <b>NAND_HandleTypeDef * hnand)</b>
Function Description	return the NAND state
Parameters	• <b>hnand</b> : pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	• <b>HAL state</b>
Notes	• None.

## 30.3 NAND Firmware driver defines

### 30.3.1 NAND

NAND

*NAND\_Exported\_Constants*

- #define: **NAND\_DEVICE1 ((uint32\_t)0x70000000)**
- #define: **NAND\_DEVICE2 ((uint32\_t)0x80000000)**
- #define: **NAND\_WRITE\_TIMEOUT ((uint32\_t)0x01000000)**
- #define: **CMD\_AREA ((uint32\_t)(1<<16))**

- #define: **ADDR\_AREA** ((*uint32\_t*)*(1<<17)*)
- #define: **NAND\_CMD\_AREA\_A** ((*uint8\_t*)*0x00*)
- #define: **NAND\_CMD\_AREA\_B** ((*uint8\_t*)*0x01*)
- #define: **NAND\_CMD\_AREA\_C** ((*uint8\_t*)*0x50*)
- #define: **NAND\_VALID\_ADDRESS** ((*uint32\_t*)*0x00000100*)
- #define: **NAND\_INVALID\_ADDRESS** ((*uint32\_t*)*0x00000200*)
- #define: **NAND\_TIMEOUT\_ERROR** ((*uint32\_t*)*0x00000400*)
- #define: **NAND\_BUSY** ((*uint32\_t*)*0x00000000*)
- #define: **NAND\_ERROR** ((*uint32\_t*)*0x00000001*)
- #define: **NAND\_READY** ((*uint32\_t*)*0x00000040*)

## 31 HAL NOR Generic Driver

### 31.1 NOR Firmware driver registers structures

#### 31.1.1 NOR\_HandleTypeDef

*NOR\_HandleTypeDef* is defined in the `stm32f4xx_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

#### 31.1.2 NOR\_CFITypeDef

*NOR\_CFITypeDef* is defined in the `stm32f4xx_hal_nor.h`

##### Data Fields

- *uint16\_t CFI\_1*
- *uint16\_t CFI\_2*
- *uint16\_t CFI\_3*
- *uint16\_t CFI\_4*

##### Field Documentation

- *uint16\_t NOR\_CFITypeDef::CFI\_1*
  - < Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory

- *uint16\_t NOR\_CFITypeDef::CFI\_2*
- *uint16\_t NOR\_CFITypeDef::CFI\_3*
- *uint16\_t NOR\_CFITypeDef::CFI\_4*

### 31.1.3 NOR\_IDTypeDef

*NOR\_IDTypeDef* is defined in the `stm32f4xx_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 devices' 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

## 31.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

### 31.2.1 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/FSMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function `HAL_NOR_Init()` with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function `HAL_NOR_Read_ID()`. The read information is stored in the `NOR_ID_TypeDef` structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions `HAL_NOR_Read()`, `HAL_NOR_Program()`.
- Perform NOR flash erase block/chip operations using the functions `HAL_NOR_Erase_Block()` and `HAL_NOR_Erase_Chip()`.
- Read the NOR flash CFI (common flash interface) IDs using the function `HAL_NOR_Read_CFI()`. The read information is stored in the `NOR_CFI_TypeDef` structure declared by the function caller.

- You can also control the NOR device by calling the control APIs `HAL_NOR_WriteOperation_Enable()`/ `HAL_NOR_WriteOperation_Disable()` to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function `HAL_NOR_GetState()`



This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

### NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- `_NOR_WRITE` : NOR memory write data to specified address

#### 31.2.2 NOR Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

- `HAL_NOR_Init()`
- `HAL_NOR_DelInit()`
- `HAL_NOR_MspInit()`
- `HAL_NOR_MspDelInit()`
- `HAL_NOR_MspWait()`

#### 31.2.3 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()`

#### 31.2.4 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()`

#### 31.2.5 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\(\)\*](#)

### 31.2.6 Initialization and de-initialization functions

#### 31.2.6.1 HAL\_NOR\_Init

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Init ( <i>NOR_HandleTypeDef</i> * <i>hnor</i>, <i>FMC_NORSRAM_TimingTypeDef</i> * <i>Timing</i>, <i>FMC_NORSRAM_TimingTypeDef</i> * <i>ExtTiming</i>)</b>
Function Description	Perform the NOR memory Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>Timing</b> : pointer to NOR control timing structure</li> <li>• <b>ExtTiming</b> : pointer to NOR extended mode timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 31.2.6.2 HAL\_NOR\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_NOR_DeInit ( <i>NOR_HandleTypeDef</i> * <i>hnor</i>)</b>
Function Description	Perform NOR memory De-Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

Notes

#### 31.2.6.3 HAL\_NOR\_MspInit

Function Name	<b>void HAL_NOR_MspInit ( <i>NOR_HandleTypeDef</i> * <i>hnor</i>)</b>
---------------	---

Function Description	NOR MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 31.2.6.4 HAL\_NOR\_MspDeInit

Function Name	<b>void HAL_NOR_MspDeInit ( NOR_HandleTypeDef * hnor)</b>
Function Description	NOR MSP DeInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 31.2.6.5 HAL\_NOR\_MspWait

Function Name	<b>void HAL_NOR_MspWait ( NOR_HandleTypeDef * hnor, uint32_t Timeout)</b>
Function Description	NOR BSP Wait fro Ready/Busy signal.
Parameters	<ul style="list-style-type: none"><li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li><li>• <b>Timeout</b> : Maximum timeout value</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 31.2.7 Input and Output functions

### 31.2.7.1 HAL\_NOR\_Read\_ID

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Read_ID (</b> <b>NOR_HandleTypeDef * hnор, NOR_IDTypeDef * pNOR_ID)</b>
Function Description	Read NOR flash IDs.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnор</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>pNOR_ID</b> : pointer to NOR ID structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 31.2.7.2 HAL\_NOR\_ReturnToReadMode

Function Name	<b>HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (</b> <b>NOR_HandleTypeDef * hnор)</b>
Function Description	Returns the NOR memory to Read mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnор</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 31.2.7.3 HAL\_NOR\_Read

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Read (</b> <b>NOR_HandleTypeDef * hnор, uint32_t * pAddress, uint16_t * pData)</b>
Function Description	Read data from NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnор</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>pAddress</b> : pointer to Device address</li> <li>• <b>pData</b> : pointer to read data</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 31.2.7.4 HAL\_NOR\_Program

Function Name	<code>HAL_StatusTypeDef HAL_NOR_Program (</code> <code>NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t *</code> <code>pData)</code>
Function Description	Program data to NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>pAddress</b> : Device address</li> <li>• <b>pData</b> : pointer to the data to write</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 31.2.7.5 HAL\_NOR\_ReadBuffer

Function Name	<code>HAL_StatusTypeDef HAL_NOR_ReadBuffer (</code> <code>NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t *</code> <code>pData, uint32_t uwBufferSize)</code>
Function Description	Reads a block of data from the FMC NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>uwAddress</b> : NOR memory internal address to read from.</li> <li>• <b>pData</b> : pointer to the buffer that receives the data read from the NOR memory.</li> <li>• <b>uwBufferSize</b> : number of Half word to read.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 31.2.7.6 HAL\_NOR\_ProgramBuffer

Function Name	<code>HAL_StatusTypeDef HAL_NOR_ProgramBuffer (</code> <code>NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t *</code>
---------------	---

	<b>pData, uint32_t uwBufferSize)</b>
Function Description	Writes a half-word buffer to the FMC NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>uwAddress</b> : NOR memory internal address from which the data</li> <li>• <b>pData</b> : pointer to source data buffer.</li> <li>• <b>uwBufferSize</b> : number of Half words to write. The maximum allowed</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 31.2.7.7 HAL\_NOR\_Erase\_Block

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Erase_Block (</b> <b>NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)</b>
Function Description	Erase the specified block of the NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>BlockAddress</b> : Block to erase address</li> <li>• <b>Address</b> : Device address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 31.2.7.8 HAL\_NOR\_Erase\_Chip

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Erase_Chip (</b> <b>NOR_HandleTypeDef * hnor, uint32_t Address)</b>
Function Description	Erase the entire NOR chip.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>Address</b> : Device address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

### 31.2.7.9 HAL\_NOR\_Read\_CFI

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Read_CFI (</b> <b>NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)</b>
Function Description	Read NOR flash CFI IDs.
Parameters	<ul style="list-style-type: none"><li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li><li>• <b>pNOR_CFI</b> : pointer to NOR CFI IDs structure</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 31.2.8 Control functions

### 31.2.8.1 HAL\_NOR\_WriteOperation\_Enable

Function Name	<b>HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (</b> <b>NOR_HandleTypeDef * hnor)</b>
Function Description	Enables dynamically NOR write operation.
Parameters	<ul style="list-style-type: none"><li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>

## Notes

### 31.2.8.2 HAL\_NOR\_WriteOperation\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (</b> <b>NOR_HandleTypeDef * hnor)</b>
---------------	---

---

Function Description	Disables dynamically NOR write operation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 31.2.9 State functions

#### 31.2.9.1 HAL\_NOR\_GetState

Function Name	<b>HAL_NOR_StateTypeDef HAL_NOR_GetState (</b> <b>NOR_HandleTypeDef * hnor)</b>
Function Description	return the NOR controller state
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>NOR controller state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 31.2.9.2 HAL\_NOR\_GetStatus

Function Name	<b>NOR_StatusTypeDef HAL_NOR_GetStatus (</b> <b>NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)</b>
Function Description	Returns the NOR operation status.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor</b> : pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>Address</b> : Device address</li> <li>• <b>Timeout</b> : NOR progamming Timeout</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>NOR_Status : The returned value can be:</b> <b>NOR_SUCCESS, NOR_ERROR or NOR_TIMEOUT</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 31.3 NOR Firmware driver defines

### 31.3.1 NOR

NOR

*NOR\_Exported\_Constants*

- #define: **MC\_ADDRESS** ((*uint16\_t*)0x0000)
- #define: **DEVICE\_CODE1\_ADDR** ((*uint16\_t*)0x0001)
- #define: **DEVICE\_CODE2\_ADDR** ((*uint16\_t*)0x000E)
- #define: **DEVICE\_CODE3\_ADDR** ((*uint16\_t*)0x000F)
- #define: **CFI1\_ADDRESS** ((*uint16\_t*)0x61)
- #define: **CFI2\_ADDRESS** ((*uint16\_t*)0x62)
- #define: **CFI3\_ADDRESS** ((*uint16\_t*)0x63)
- #define: **CFI4\_ADDRESS** ((*uint16\_t*)0x64)
- #define: **NOR\_TMEOUT** ((*uint16\_t*)0xFFFF)
- #define: **NOR\_MEMORY\_8B**

- #define: **NOR\_MEMORY\_ADDRESS ((uint32\_t)0x60000000)**

## 32 HAL PCCARD Generic Driver

### 32.1 PCCARD Firmware driver registers structures

#### 32.1.1 PCCARD\_HandleTypeDef

*PCCARD\_HandleTypeDef* is defined in the `stm32f4xx_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

### 32.2 PCCARD Firmware driver API description

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

#### 32.2.1 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/FSMC layer functions to interface with PCCARD devices. This driver is used for:

- PCCARD/compact flash memory configuration sequence using the function `HAL_PCCARD_Init()` with control and timing parameters for both common and attribute spaces.
- Read PCCARD/compact flash memory maker and device IDs using the function `HAL_CF_Read_ID()`. The read information is stored in the `CompactFlash_ID` structure declared by the function caller.
- Access PCCARD/compact flash memory by read/write operations using the functions `HAL_CF_Read_Sector()`/`HAL_CF_Write_Sector()`, to read/write sector.
- Perform PCCARD/compact flash Reset chip operation using the function `HAL_CF_Reset()`.
- Perform PCCARD/compact flash erase sector operation using the function `HAL_CF_Erase_Sector()`.

- Read the PCCARD/compact flash status operation using the function `HAL_CF_ReadStatus()`.
- You can monitor the PCCARD/compact flash device HAL state by calling the function `HAL_PCCARD_GetState()`



This driver is a set of generic APIs which handle standard PCCARD/compact flash operations. If a PCCARD/compact flash device contains different operations and/or implementations, it should be implemented separately.

### 32.2.2 PCCARD Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the PCCARD memory

- `HAL_PCCARD_Init()`
- `HAL_PCCARD_DelInit()`
- `HAL_PCCARD_MspInit()`
- `HAL_PCCARD_MspDelInit()`

### 32.2.3 PCCARD Input and Output 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()`

### 32.2.4 PCCARD 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()`

### 32.2.5 Initialization and de-initialization functions

#### 32.2.5.1 `HAL_PCCARD_Init`

Function Name	<code>HAL_StatusTypeDef HAL_PCCARD_Init (</code> <code>PCCARD_HandleTypeDef * hpccard,</code> <code>FMC_NAND_PCC_TimingTypeDef * ComSpaceTiming,</code> <code>FMC_NAND_PCC_TimingTypeDef * AttSpaceTiming,</code> <code>FMC_NAND_PCC_TimingTypeDef * IOSpaceTiming)</code>
---------------	--

Function Description	Perform the PCCARD memory Initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> <li>• <b>ComSpaceTiming</b> : Common space timing structure</li> <li>• <b>AttSpaceTiming</b> : Attribute space timing structure</li> <li>• <b>IOSpaceTiming</b> : IO space timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.5.2 HAL\_PCCARD\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_PCCARD_DelInit ( <i>PCCARD_HandleTypeDef</i> * hpccard)</b>
Function Description	Perform the PCCARD memory De-initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.5.3 HAL\_PCCARD\_MspInit

Function Name	<b>void HAL_PCCARD_MspInit ( <i>PCCARD_HandleTypeDef</i> * hpccard)</b>
Function Description	PCCARD MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.5.4 HAL\_PCCARD\_MspDelInit

Function Name	<code>void HAL_PCCARD_MspDelInit ( <i>PCCARD_HandleTypeDef</i> * hpccard)</code>
Function Description	PCCARD MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 32.2.6 Input and Output functions

### 32.2.6.1 HAL\_CF\_Read\_ID

Function Name	<code>HAL_StatusTypeDef HAL_CF_Read_ID ( <i>PCCARD_HandleTypeDef</i> * hpccard, uint8_t CompactFlash_ID, uint8_t * pStatus)</code>
Function Description	Read Compact Flash's ID.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> <li>• <b>CompactFlash_ID</b> : Compact flash ID structure.</li> <li>• <b>pStatus</b> : pointer to compact flash status</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.6.2 HAL\_CF\_Read\_Sector

Function Name	<code>HAL_StatusTypeDef HAL_CF_Read_Sector ( <i>PCCARD_HandleTypeDef</i> * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)</code>
---------------	---

---

Function Description	Read sector from PCCARD memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> <li>• <b>pBuffer</b> : pointer to destination read buffer</li> <li>• <b>SectorAddress</b> : Sector address to read</li> <li>• <b>pStatus</b> : pointer to CF status</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.6.3 HAL\_CF\_Write\_Sector

Function Name	<code>HAL_StatusTypeDef HAL_CF_Write_Sector ( PCCARD_HandleTypeDef * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)</code>
Function Description	Write sector to PCCARD memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> <li>• <b>pBuffer</b> : pointer to source write buffer</li> <li>• <b>SectorAddress</b> : Sector address to write</li> <li>• <b>pStatus</b> : pointer to CF status</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.6.4 HAL\_CF\_Erase\_Sector

Function Name	<code>HAL_StatusTypeDef HAL_CF_Erase_Sector ( PCCARD_HandleTypeDef * hpccard, uint16_t SectorAddress, uint8_t * pStatus)</code>
Function Description	Erase sector from PCCARD memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> <li>• <b>SectorAddress</b> : Sector address to erase</li> </ul>

- **pStatus** : pointer to CF status
  - **HAL status**
  - None.
- Return values
- Notes

### 32.2.6.5 HAL\_CF\_Reset

Function Name	<b>HAL_StatusTypeDef HAL_CF_Reset (</b> <b>PCCARD_HandleTypeDef * hpccard)</b>
Function Description	Reset the PCCARD memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.6.6 HAL\_PCCARD\_IRQHandler

Function Name	<b>void HAL_PCCARD_IRQHandler (</b> <b>PCCARD_HandleTypeDef * hpccard)</b>
Function Description	This function handles PCCARD device interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.6.7 HAL\_PCCARD\_ITCallback

Function Name	<b>void HAL_PCCARD_ITCallback ( <i>PCCARD_HandleTypeDef</i>* hpccard)</b>
Function Description	PCCARD interrupt feature callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 32.2.7 State functions

#### 32.2.7.1 HAL\_PCCARD\_GetState

Function Name	<b>HAL_PCCARD_StateTypeDef HAL_PCCARD_GetState ( <i>PCCARD_HandleTypeDef</i>* hpccard)</b>
Function Description	return the PCCARD controller state
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 32.2.7.2 HAL\_CF\_GetStatus

Function Name	<b>CF_StatusTypeDef HAL_CF_GetStatus ( <i>PCCARD_HandleTypeDef</i>* hpccard)</b>
Function Description	Get the compact flash memory status.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>New status of the CF operation. This parameter can be:</b> <ul style="list-style-type: none"> <li>– <b>CompactFlash_TIMEOUT_ERROR: when the previous operation generate a Timeout error</b></li> <li>– <b>CompactFlash_READY: when memory is ready for</b></li> </ul> </li> </ul>

*the next operation*

## Notes

- None.

**32.2.7.3 HAL\_CF\_ReadStatus**

Function Name	<b>CF_StatusTypeDef HAL_CF_ReadStatus (</b> <b>PCCARD_HandleTypeDef * hpccard)</b>
Function Description	Reads the Compact Flash memory status using the Read status command.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpccard</b> : pointer to a PCCARD_HandleTypeDef structure that contains the configuration information for PCCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>The status of the Compact Flash memory. This parameter can be:</b> <ul style="list-style-type: none"> <li>– <b>CompactFlash_BUSY: when memory is busy</b></li> <li>– <b>CompactFlash_READY: when memory is ready for the next operation</b></li> <li>– <b>CompactFlash_ERROR: when the previous operation generates error</b></li> </ul> </li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**32.3 PCCARD Firmware driver defines****32.3.1 PCCARD**

PCCARD

***PCCARD\_Exported\_Constants***

- #define: **CF\_DEVICE\_ADDRESS ((uint32\_t)0x90000000)**
- #define: **CF\_ATTRIBUTE\_SPACE\_ADDRESS ((uint32\_t)0x98000000)**
- #define: **CF\_COMMON\_SPACE\_ADDRESS CF\_DEVICE\_ADDRESS**

- #define: ***CF\_IO\_SPACE\_ADDRESS*** ((*uint32\_t*)0x9C000000)
- #define: ***CF\_IO\_SPACE\_PRIMARY\_ADDR*** ((*uint32\_t*)0x9C0001F0)
- #define: ***CF\_DATA*** ((*uint8\_t*)0x00)
- #define: ***CF\_SECTOR\_COUNT*** ((*uint8\_t*)0x02)
- #define: ***CF\_SECTOR\_NUMBER*** ((*uint8\_t*)0x03)
- #define: ***CF\_CYLINDER\_LOW*** ((*uint8\_t*)0x04)
- #define: ***CF\_CYLINDER\_HIGH*** ((*uint8\_t*)0x05)
- #define: ***CF\_CARD\_HEAD*** ((*uint8\_t*)0x06)
- #define: ***CF\_STATUS\_CMD*** ((*uint8\_t*)0x07)
- #define: ***CF\_STATUS\_CMD\_ALTERNATE*** ((*uint8\_t*)0x0E)
- #define: ***CF\_COMMON\_DATA\_AREA*** ((*uint16\_t*)0x0400)
- #define: ***CF\_READ\_SECTOR\_CMD*** ((*uint8\_t*)0x20)

- #define: ***CF\_WRITE\_SECTOR\_CMD*** ((*uint8\_t*)0x30)
- #define: ***CF\_ERASE\_SECTOR\_CMD*** ((*uint8\_t*)0xC0)
- #define: ***CF\_IDENTIFY\_CMD*** ((*uint8\_t*)0xEC)
- #define: ***CF\_TIMEOUT\_ERROR*** ((*uint8\_t*)0x60)
- #define: ***CF\_BUSY*** ((*uint8\_t*)0x80)
- #define: ***CF\_PROGR*** ((*uint8\_t*)0x01)
- #define: ***CF\_READY*** ((*uint8\_t*)0x40)
- #define: ***CF\_SECTOR\_SIZE*** ((*uint32\_t*)255)

## 33 HAL PCD Generic Driver

### 33.1 PCD Firmware driver registers structures

#### 33.1.1 PCD\_HandleTypeDef

*PCD\_HandleTypeDef* is defined in the `stm32f4xx_hal_pcd.h`

##### Data Fields

- *PCD\_TypeDef \* Instance*
- *PCD\_InitTypeDef Init*
- *PCD\_EPTTypeDef IN\_ep*
- *PCD\_EPTTypeDef OUT\_ep*
- *HAL\_LockTypeDef Lock*
- *\_\_IO PCD\_StateTypeDef State*
- *uint32\_t Setup*
- *void \* pData*

##### Field Documentation

- *PCD\_TypeDef\* PCD\_HandleTypeDef::Instance*
  - Register base address
- *PCD\_InitTypeDef PCD\_HandleTypeDef::Init*
  - PCD required parameters
- *PCD\_EPTTypeDef PCD\_HandleTypeDef::IN\_ep[15]*
  - IN endpoint parameters
- *PCD\_EPTTypeDef PCD\_HandleTypeDef::OUT\_ep[15]*
  - OUT endpoint parameters
- *HAL\_LockTypeDef PCD\_HandleTypeDef::Lock*
  - PCD peripheral status
- *\_\_IO PCD\_StateTypeDef PCD\_HandleTypeDef::State*
  - PCD communication state
- *uint32\_t PCD\_HandleTypeDef::Setup[12]*
  - Setup packet buffer
- *void\* PCD\_HandleTypeDef::pData*
  - Pointer to upper stack Handler

### 33.2 PCD Firmware driver API description

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

#### 33.2.1 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
    - \_\_OTGFS-OTG\_CLK\_ENABLE() / \_\_OTGHS-OTG\_CLK\_ENABLE();
    - \_\_OTGHSULPI\_CLK\_ENABLE(); (For High Speed Mode)
  - b. Initialize the related GPIO clocks
  - c. Configure PCD pin-out
  - d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
  - a. hpcd.pData = pdev;
6. Enable HCD transmission and reception:
  - a. HAL\_PCD\_Start();

### 33.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- [\*HAL\\_PCD\\_Init\(\)\*](#)
- [\*HAL\\_PCD\\_DelInit\(\)\*](#)
- [\*HAL\\_PCD\\_MspInit\(\)\*](#)
- [\*HAL\\_PCD\\_MspDelInit\(\)\*](#)

### 33.2.3 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\(\)\*](#)

### 33.2.4 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\\_SetTxFifo\(\)\*](#)
- [\*HAL\\_PCD\\_SetRxFifo\(\)\*](#)
- [\*HAL\\_PCD\\_ActiveRemoteWakeUp\(\)\*](#)
- [\*HAL\\_PCD\\_DeActiveRemoteWakeUp\(\)\*](#)

### 33.2.5 Peripheral State functions

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

- [\*HAL\\_PCD\\_GetState\(\)\*](#)

### 33.2.6 Initialization and de-initialization functions

#### 33.2.6.1 HAL\_PCD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Init ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Initializes the PCD according to the specified parameters in the <i>PCD_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 33.2.6.2 HAL\_PCD\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DelInit ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Deinitializes the PCD peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.6.3 HAL\_PCD\_MspInit

Function Name	<b>void HAL_PCD_MspInit ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Initializes the PCD MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.6.4 HAL\_PCD\_MspDeInit

Function Name	<b>void HAL_PCD_MspDeInit ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Deinitializes PCD MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 33.2.7 IO operation functions

### 33.2.7.1 HAL\_PCD\_Start

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Start ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Start The USB OTG Device.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.2 HAL\_PCD\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Stop ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Stop The USB OTG Device.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.3 HAL\_PCD\_IRQHandler

Function Name	<b>void HAL_PCD_IRQHandler ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	This function handles PCD interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.4 HAL\_PCD\_DataOutStageCallback

Function Name	<b>void HAL_PCD_DataOutStageCallback ( <i>PCD_HandleTypeDef</i> * hpcd, uint8_t epxnum)</b>
Function Description	Data out stage callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.5 HAL\_PCD\_DataInStageCallback

Function Name	<b>void HAL_PCD_DataInStageCallback ( <i>PCD_HandleTypeDef</i> * <i>hpcd</i>, <i>uint8_t</i> <i>epnum</i>)</b>
Function Description	Data IN stage callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b><i>hpcd</i></b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.6 HAL\_PCD\_SetupStageCallback

Function Name	<b>void HAL_PCD_SetupStageCallback ( <i>PCD_HandleTypeDef</i> * <i>hpcd</i>)</b>
Function Description	Setup stage callback.
Parameters	<ul style="list-style-type: none"><li>• <b><i>hpcd</i></b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.7 HAL\_PCD\_SOFCallback

Function Name	<b>void HAL_PCD_SOFCallback ( <i>PCD_HandleTypeDef</i> * <i>hpcd</i>)</b>
Function Description	USB Start Of Frame callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b><i>hpcd</i></b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.8 HAL\_PCD\_ResetCallback

Function Name	<b>void HAL_PCD_ResetCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	USB Reset callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.9 HAL\_PCD\_SuspendCallback

Function Name	<b>void HAL_PCD_SuspendCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Suspend event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.10 HAL\_PCD\_ResumeCallback

Function Name	<b>void HAL_PCD_ResumeCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Resume event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.11 HAL\_PCD\_ISOOUTIncompleteCallback

Function Name	<b>void HAL_PCD_ISOOUTIncompleteCallback ( <i>PCD_HandleTypeDef</i> * hpcd, uint8_t epcnum)</b>
Function Description	Incomplete ISO OUT callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.12 HAL\_PCD\_ISOINIncompleteCallback

Function Name	<b>void HAL_PCD_ISOINIncompleteCallback ( <i>PCD_HandleTypeDef</i> * hpcd, uint8_t epcnum)</b>
Function Description	Incomplete ISO IN callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.13 HAL\_PCD\_ConnectCallback

Function Name	<b>void HAL_PCD_ConnectCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Connection event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.7.14 HAL\_PCD\_DisconnectCallback

Function Name	<b>void HAL_PCD_DisconnectCallback ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Disconnection event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 33.2.8 Peripheral Control functions

### 33.2.8.1 HAL\_PCD\_DevConnect

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DevConnect ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>

### 33.2.8.2 HAL\_PCD\_DevDisconnect

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DevDisconnect ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>

### 33.2.8.3 HAL\_PCD\_SetAddress

Function Name	<b>HAL_StatusTypeDef HAL_PCD_SetAddress ( PCD_HandleTypeDef * hpcd, uint8_t address)</b>
Function Description	Set the USB Device address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> <li>• <b>address</b> : new device address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.8.4 HAL\_PCD\_EP\_Open

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Open ( PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)</b>
Function Description	Open and configure an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> <li>• <b>ep_addr</b> : endpoint address</li> <li>• <b>ep_mps</b> : endpoint max packet size</li> <li>• <b>ep_type</b> : endpoint type</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.8.5 HAL\_PCD\_EP\_Close

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Close ( PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Deactivate an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> <li>• <b>ep_addr</b> : endpoint address</li> </ul>

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>             |

### 33.2.8.6 HAL\_PCD\_EP\_Receive

Function Name	<code>HAL_StatusTypeDef HAL_PCD_EP_Receive (</code> <code>    PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf,</code> <code>    uint32_t len)</code>
Function Description	Receive an amount of data.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li><li>• <b>ep_addr</b> : endpoint address</li><li>• <b>pBuf</b> : pointer to the reception buffer</li><li>• <b>len</b> : amount of data to be received</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.8.7 HAL\_PCD\_EP\_GetRxCount

Function Name	<code>uint16_t HAL_PCD_EP_GetRxCount (</code> <code>    PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</code>
Function Description	Get Received Data Size.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li><li>• <b>ep_addr</b> : endpoint address</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>Data Size</b></li></ul>

### 33.2.8.8 HAL\_PCD\_EP\_Transmit

---

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Transmit (</b> <i>PCD_HandleTypeDef</i> * <b>hpcd</b> , uint8_t <b>ep_addr</b> , uint8_t * <b>pBuf</b> , uint32_t <b>len</b> )
Function Description	Send an amount of data.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> <li>• <b>ep_addr</b> : endpoint address</li> <li>• <b>pBuf</b> : pointer to the transmission buffer</li> <li>• <b>len</b> : amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.8.9 HAL\_PCD\_EP\_SetStall

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_SetStall (</b> <i>PCD_HandleTypeDef</i> * <b>hpcd</b> , uint8_t <b>ep_addr</b> )
Function Description	Set a STALL condition over an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> <li>• <b>ep_addr</b> : endpoint address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

### 33.2.8.10 HAL\_PCD\_EP\_ClrStall

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_ClrStall (</b> <i>PCD_HandleTypeDef</i> * <b>hpcd</b> , uint8_t <b>ep_addr</b> )
Function Description	Clear a STALL condition over in an endpoint.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> <li>• <b>ep_addr</b> : endpoint address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

Notes

### 33.2.8.11 HAL\_PCD\_EP\_Flush

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Flush (</b> <b>PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
Function Description	Flush an endpoint.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li><li>• <b>ep_addr</b> : endpoint address</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 33.2.8.12 HAL\_PCD\_SetTxFiFo

Function Name	<b>HAL_StatusTypeDef HAL_PCD_SetTxFiFo (</b> <b>PCD_HandleTypeDef * hpcd, uint8_t fifo, uint16_t size)</b>
Function Description	Update FIFO configuration.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>

### 33.2.8.13 HAL\_PCD\_SetRxFiFo

Function Name	<b>HAL_StatusTypeDef HAL_PCD_SetRxFiFo (</b> <b>PCD_HandleTypeDef * hpcd, uint16_t size)</b>
Function Description	Update FIFO configuration.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b> : PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>

### 33.2.8.14 HAL\_PCD\_ActiveRemoteWakeup

Function Name	<b>HAL_StatusTypeDef HAL_PCD_ActiveRemoteWakeup ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	HAL_PCD_ActiveRemoteWakeup : active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.2.8.15 HAL\_PCD\_DeActiveRemoteWakeup

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DeActiveRemoteWakeup ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	HAL_PCD_DeActiveRemoteWakeup : de-active remote wakeup signalling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 33.2.9 Peripheral State functions

### 33.2.9.1 HAL\_PCD\_GetState

Function Name	<b>PCD_StateTypeDef HAL_PCD_GetState ( <i>PCD_HandleTypeDef</i> * hpcd)</b>
Function Description	Return the PCD state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b> : PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 33.3 PCD Firmware driver defines

#### 33.3.1 PCD

PCD

*PCD\_Interrupt\_Clock*

- #define: ***USB\_FS EXTI\_TRIGGER\_RISING\_EDGE ((uint32\_t)0x08)***
- #define: ***USB\_FS EXTI\_TRIGGER\_FALLING\_EDGE ((uint32\_t)0x0C)***
- #define: ***USB\_FS EXTI\_TRIGGER\_BOTH\_EDGE ((uint32\_t)0x10)***
- #define: ***USB\_HS EXTI\_TRIGGER\_RISING\_EDGE ((uint32\_t)0x08)***
- #define: ***USB\_HS EXTI\_TRIGGER\_FALLING\_EDGE ((uint32\_t)0x0C)***
- #define: ***USB\_HS EXTI\_TRIGGER\_BOTH\_EDGE ((uint32\_t)0x10)***
- #define: ***USB\_HS EXTI\_LINE\_WAKEUP ((uint32\_t)0x00100000)***  
*External interrupt line 20 Connected to the USB HS EXTI Line*
- #define: ***USB\_FS EXTI\_LINE\_WAKEUP ((uint32\_t)0x00040000)***  
*External interrupt line 18 Connected to the USB FS EXTI Line*
- #define: ***\_\_HAL\_USB\_HS\_EXTI\_ENABLE\_IT EXTI->IMR |= (USB\_HS\_EXTI\_LINE\_WAKEUP)***
- #define: ***\_\_HAL\_USB\_HS\_EXTI\_DISABLE\_IT EXTI->IMR &= ~(USB\_HS\_EXTI\_LINE\_WAKEUP)***

- #define: **`__HAL_USB_HS_EXTI_GET_FLAG EXTI->PR & (USB_HS_EXTI_LINE_WAKEUP)`**
  
- #define: **`__HAL_USB_HS_EXTI_CLEAR_FLAG EXTI->PR = (USB_HS_EXTI_LINE_WAKEUP)`**
  
- #define: **`__HAL_USB_HS_EXTI_SET_RISING_EDGE_TRIGGER EXTI->FTSR &= ~(USB_HS_EXTI_LINE_WAKEUP); \ EXTI->RTSR |= USB_HS_EXTI_LINE_WAKEUP`**
  
- #define: **`__HAL_USB_HS_EXTI_SET_FALLING_EDGE_TRIGGER EXTI->FTSR |= (USB_HS_EXTI_LINE_WAKEUP); \ EXTI->RTSR &= ~(USB_HS_EXTI_LINE_WAKEUP)`**
  
- #define: **`__HAL_USB_HS_EXTI_SET_FALLINGRISING_TRIGGER EXTI->RTSR &= ~(USB_HS_EXTI_LINE_WAKEUP); \ EXTI->FTSR &= ~(USB_HS_EXTI_LINE_WAKEUP); \ EXTI->RTSR |= USB_HS_EXTI_LINE_WAKEUP; \ EXTI->FTSR |= USB_HS_EXTI_LINE_WAKEUP`**
  
- #define: **`__HAL_USB_FS_EXTI_ENABLE_IT EXTI->IMR |= USB_FS_EXTI_LINE_WAKEUP`**
  
- #define: **`__HAL_USB_FS_EXTI_DISABLE_IT EXTI->IMR &= ~(USB_FS_EXTI_LINE_WAKEUP)`**
  
- #define: **`__HAL_USB_FS_EXTI_GET_FLAG EXTI->PR & (USB_FS_EXTI_LINE_WAKEUP)`**
  
- #define: **`__HAL_USB_FS_EXTI_CLEAR_FLAG EXTI->PR = USB_FS_EXTI_LINE_WAKEUP`**

- #define: \_\_HAL\_USB\_FS\_EXTI\_SET\_RISING\_EDGE\_TRIGGER EXTI->FTSR &= ~(**USB\_FS\_EXTI\_LINE\_WAKEUP**); \ EXTI->RTSR != **USB\_FS\_EXTI\_LINE\_WAKEUP**
- #define: \_\_HAL\_USB\_FS\_EXTI\_SET\_FALLING\_EDGE\_TRIGGER EXTI->FTSR != (**USB\_FS\_EXTI\_LINE\_WAKEUP**); \ EXTI->RTSR &= ~(**USB\_FS\_EXTI\_LINE\_WAKEUP**)
- #define: \_\_HAL\_USB\_FS\_EXTI\_SET\_FALLINGRISING\_TRIGGER EXTI->RTSR &= ~(**USB\_FS\_EXTI\_LINE\_WAKEUP**); \ EXTI->FTSR &= ~(**USB\_FS\_EXTI\_LINE\_WAKEUP**); \ EXTI->RTSR != **USB\_FS\_EXTI\_LINE\_WAKEUP**; \ EXTI->FTSR != **USB\_FS\_EXTI\_LINE\_WAKEUP**

#### *PCD\_PHY\_Module*

- #define: **PCD\_PHY\_ULPI** 1
- #define: **PCD\_PHY\_EMBEDDED** 2

#### *PCD\_Speed*

- #define: **PCD\_SPEED\_HIGH** 0
- #define: **PCD\_SPEED\_HIGH\_IN\_FULL** 1
- #define: **PCD\_SPEED\_FULL** 2

## 34 HAL PWR Generic Driver

### 34.1 PWR Firmware driver registers structures

#### 34.1.1 PWR\_PVDTTypeDef

*PWR\_PVDTTypeDef* is defined in the `stm32f4xx_hal_pwr.h`

##### Data Fields

- *uint32\_t PVDLevel*
- *uint32\_t Mode*

##### Field Documentation

- *uint32\_t PWR\_PVDTTypeDef::PVDLevel*
  - PVDLevel: Specifies the PVD detection level. This parameter can be a value of [\*PWR\\_PVD\\_detection\\_level\*](#)
- *uint32\_t PWR\_PVDTTypeDef::Mode*
  - Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [\*PWR\\_PVD\\_Mode\*](#)

#### 34.1.2 PWR\_TypeDef

*PWR\_TypeDef* is defined in the `stm32f439xx.h`

##### Data Fields

- *\_\_IO uint32\_t CR*
- *\_\_IO uint32\_t CSR*

##### Field Documentation

- *\_\_IO uint32\_t PWR\_TypeDef::CR*
  - PWR power control register, Address offset: 0x00
- *\_\_IO uint32\_t PWR\_TypeDef::CSR*
  - PWR power control/status register, Address offset: 0x04

### 34.2 PWR Firmware driver API description

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

#### 34.2.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses. To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__PWR_CLK_ENABLE()` macro.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.
- `HAL_PWR_DelInit()`
- `HAL_PWR_EnableBkUpAccess()`
- `HAL_PWR_DisableBkUpAccess()`

### 34.2.2 Peripheral Control functions

#### PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the PWR\_CR).
- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through `__HAL_PVD_EXTI_ENABLE_IT()` macro.
- The PVD is stopped in Standby mode.

#### WakeUp pin configuration

- WakeUp pin is used to wake up the system from Standby mode. This pin is forced in input pull-down configuration and is active on rising edges.
- There is only one WakeUp pin: WakeUp Pin 1 on PA.00.

#### 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.

#### Sleep mode

- Entry: The Sleep mode is entered by using the `HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI)` functions with
  - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
  - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction. The Regulator parameter is not used for the STM32F4 family and is kept as parameter just to maintain compatibility with the lower power families (STM32L).
- Exit: Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

## Stop mode

In Stop mode, all clocks in the 1.2V domain are stopped, the PLL, the HSI, and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. The voltage regulator can be configured either in normal or low-power mode. To minimize the consumption In Stop mode, FLASH can be powered off before entering the Stop mode using the HAL\_PWR\_EnableFlashPowerDown() function. It can be switched on again by software after exiting the Stop mode using the HAL\_PWR\_DisableFlashPowerDown() function.

- Entry: The Stop mode is entered using the HAL\_PWR\_EnterSTOPMode(PWR\_MAINREGULATOR\_ON) function with:
  - Main regulator ON.
  - Low Power regulator ON.
- Exit: Any EXTI Line (Internal or External) configured in Interrupt/Event mode.

## 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 1.2V 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 or a time-stamp event, without depending on an external interrupt (Auto-wakeup mode).
- RTC auto-wakeup (AWU) from the Stop and Standby modes
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL\_RTC\_SetAlarm\_IT() function.
  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL\_RTCEx\_SetTimeStamp\_IT() or HAL\_RTCEx\_SetTamper\_IT() functions.
  - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL\_RTCEx\_SetWakeUpTimer\_IT() function.
- [\*\*HAL\\_PWR\\_PVDConfig\(\)\*\*](#)
- [\*\*HAL\\_PWR\\_EnablePVD\(\)\*\*](#)
- [\*\*HAL\\_PWR\\_DisablePVD\(\)\*\*](#)
- [\*\*HAL\\_PWR\\_EnableWakeUpPin\(\)\*\*](#)
- [\*\*HAL\\_PWR\\_DisableWakeUpPin\(\)\*\*](#)
- [\*\*HAL\\_PWR\\_EnterSLEEPMode\(\)\*\*](#)
- [\*\*HAL\\_PWR\\_EnterSTOPMode\(\)\*\*](#)

- *HAL\_PWR\_EnterSTANDBYMode()*
- *HAL\_PWR\_PVD\_IRQHandler()*
- *HAL\_PWR\_PVDCallback()*

### 34.2.3 Initialization and de-initialization functions

#### 34.2.3.1 HAL\_PWR\_DeInit

Function Name	<b>void HAL_PWR_DeInit ( void )</b>
Function Description	Deinitializes the HAL PWR peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 34.2.3.2 HAL\_PWR\_EnableBkUpAccess

Function Name	<b>void HAL_PWR_EnableBkUpAccess ( void )</b>
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.</li></ul>

#### 34.2.3.3 HAL\_PWR\_DisableBkUpAccess

Function Name	<b>void HAL_PWR_DisableBkUpAccess ( void )</b>
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).
Parameters	<ul style="list-style-type: none"><li>• None.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>

- 
- |       |   |
|-------|---|
| Notes | <ul style="list-style-type: none"> <li>• If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.</li> </ul> |
|-------|---|

### 34.2.4 Peripheral Control functions

#### 34.2.4.1 HAL\_PWR\_PVDConfig

Function Name	<b>void HAL_PWR_PVDConfig ( <i>PWR_PVDTTypeDef</i> * sConfigPVD)</b>
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> <li>• <b>sConfigPVD</b> : pointer to an <i>PWR_PVDTTypeDef</i> structure that contains the configuration information for the PVD.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.</li> </ul>

#### 34.2.4.2 HAL\_PWR\_EnablePVD

Function Name	<b>void HAL_PWR_EnablePVD ( void )</b>
Function Description	Enables the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

Notes

• None.

#### 34.2.4.3 HAL\_PWR\_DisablePVD

Function Name	<b>void HAL_PWR_DisablePVD ( void )</b>
---------------	---

Function Description	Disables the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

#### 34.2.4.4 HAL\_PWR\_EnableWakeUpPin

Function Name	<b>void HAL_PWR_EnableWakeUpPin ( uint32_t WakeUpPinx)</b>
Function Description	Enables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"><li><b>WakeUpPinx</b> : Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values:<ul style="list-style-type: none"><li><b>PWR_WAKEUP_PIN1</b> :</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

#### 34.2.4.5 HAL\_PWR\_DisableWakeUpPin

Function Name	<b>void HAL_PWR_DisableWakeUpPin ( uint32_t WakeUpPinx)</b>
Function Description	Disables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"><li><b>WakeUpPinx</b> : Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:<ul style="list-style-type: none"><li><b>PWR_WAKEUP_PIN1</b> :</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

#### 34.2.4.6 HAL\_PWR\_EnterSLEEPMode

Function Name	<b>void HAL_PWR_EnterSLEEPMode ( uint32_t Regulator, uint8_t SLEEPEntry)</b>
Function Description	Enters Sleep mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Regulator</b> : Specifies the regulator state in SLEEP mode. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>PWR_MAINREGULATOR_ON</b> : SLEEP mode with regulator ON</li> <li>– <b>PWR_LOWPOWERREGULATOR_ON</b> : SLEEP mode with low power regulator ON</li> </ul> </li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>SLEEPEntry</b> : Specifies if SLEEP mode is entered with WFI or WFE instruction. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>PWR_SLEEPENTRY_WFI</b> : enter SLEEP mode with WFI instruction</li> <li>– <b>PWR_SLEEPENTRY_WFE</b> : enter SLEEP mode with WFE instruction</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In Sleep mode, all I/O pins keep the same state as in Run mode.</li> <li>• In Sleep mode, the systick is stopped to avoid exit from this mode with systick interrupt when used as time base for Timeout</li> <li>• This parameter is not used for the STM32F4 family and is kept as parameter just to maintain compatibility with the lower power families.</li> </ul>

#### 34.2.4.7 HAL\_PWR\_EnterSTOPMode

Function Name	<b>void HAL_PWR_EnterSTOPMode ( uint32_t Regulator, uint8_t STOPEntry)</b>
Function Description	Enters Stop mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>Regulator</b> : Specifies the regulator state in Stop mode. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>PWR_MAINREGULATOR_ON</b> : Stop mode with regulator ON</li> <li>– <b>PWR_LOWPOWERREGULATOR_ON</b> : Stop mode with low power regulator ON</li> </ul> </li> <li>• <b>STOPEntry</b> : Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>PWR_STOPENTRY_WFI</b> : Enter Stop mode with WFI instruction</li> <li>– <b>PWR_STOPENTRY_WFE</b> : Enter Stop mode with WFE</li> </ul> </li> </ul>

**instruction**

Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>In Stop mode, all I/O pins keep the same state as in Run mode.</li><li>When exiting Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.</li><li>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.</li></ul>

#### 34.2.4.8 HAL\_PWR\_EnterSTANDBYMode

Function Name	<b>void HAL_PWR_EnterSTANDBYMode ( void )</b>
Function Description	Enters Standby mode.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>In Standby mode, all I/O pins are high impedance except for: Reset pad (still available)RTC_AF1 pin (PC13) if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out.RTC_AF2 pin (PI8) if configured for tamper or time-stamp.WKUP pin 1 (PA0) if enabled.</li></ul>

#### 34.2.4.9 HAL\_PWR\_PVD\_IRQHandler

Function Name	<b>void HAL_PWR_PVD_IRQHandler ( void )</b>
Function Description	This function handles the PWR PVD interrupt request.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>This API should be called under the PVD_IRQHandler().</li></ul>

### 34.2.4.10 HAL\_PWR\_PVDCallback

Function Name	<b>void HAL_PWR_PVDCallback ( void )</b>
Function Description	PWR PVD interrupt callback.
Parameters	• None.
Return values	• None.
Notes	• None.

## 34.3 PWR Firmware driver defines

### 34.3.1 PWR

PWR

*PWR\_Flag*

- #define: **PWR\_FLAG\_WU PWR\_CSR\_WUF**
- #define: **PWR\_FLAG\_SB PWR\_CSR\_SBF**
- #define: **PWR\_FLAG\_PVDO PWR\_CSR\_PVDO**
- #define: **PWR\_FLAG\_BRR PWR\_CSR\_BRR**
- #define: **PWR\_FLAG\_VOSRDY PWR\_CSR\_VOSRDY**

*PWR\_PVD\_detection\_level*

- #define: **PWR\_PVDLEVEL\_0 PWR\_CR\_PLS\_LEV0**
- #define: **PWR\_PVDLEVEL\_1 PWR\_CR\_PLS\_LEV1**

- #define: **PWR\_PVDLEVEL\_2 PWR\_CR\_PLS\_LEV2**
- #define: **PWR\_PVDLEVEL\_3 PWR\_CR\_PLS\_LEV3**
- #define: **PWR\_PVDLEVEL\_4 PWR\_CR\_PLS\_LEV4**
- #define: **PWR\_PVDLEVEL\_5 PWR\_CR\_PLS\_LEV5**
- #define: **PWR\_PVDLEVEL\_6 PWR\_CR\_PLS\_LEV6**
- #define: **PWR\_PVDLEVEL\_7 PWR\_CR\_PLS\_LEV7**

#### **PWR\_PVD\_Mode**

- #define: **PWR\_MODE\_EVT ((uint32\_t)0x00000000)**  
*No Interrupt*
- #define: **PWR\_MODE\_IT\_RISING ((uint32\_t)0x00000001)**  
*External Interrupt Mode with Rising edge trigger detection*
- #define: **PWR\_MODE\_IT\_FALLING ((uint32\_t)0x00000002)**  
*External Interrupt Mode with Falling edge trigger detection*
- #define: **PWR\_MODE\_IT\_RISING\_FALLING ((uint32\_t)0x00000003)**  
*External Interrupt Mode with Rising/Falling edge trigger detection*

#### **PWR\_Regulator\_state\_in\_STOP\_mode**

- #define: **PWR\_MAINREGULATOR\_ON ((uint32\_t)0x00000000)**

- #define: **PWR\_LOWPOWERREGULATOR\_ON PWR\_CR\_LPDS**

#### **PWR\_Regulator\_Voltage\_Scale**

- #define: **PWR\_REGULATOR\_VOLTAGE\_SCALE1 ((uint32\_t)0x0000C000)**
- #define: **PWR\_REGULATOR\_VOLTAGE\_SCALE2 ((uint32\_t)0x00008000)**
- #define: **PWR\_REGULATOR\_VOLTAGE\_SCALE3 ((uint32\_t)0x00004000)**

#### **PWR\_SLEEP\_mode\_entry**

- #define: **PWR\_SLEEPENTRY\_WFI ((uint8\_t)0x01)**
- #define: **PWR\_SLEEPENTRY\_WFE ((uint8\_t)0x02)**

#### **PWR\_STOP\_mode\_entry**

- #define: **PWR\_STOPENTRY\_WFI ((uint8\_t)0x01)**
- #define: **PWR\_STOPENTRY\_WFE ((uint8\_t)0x02)**

#### **PWR\_WakeUp\_Pins**

- #define: **PWR\_WAKEUP\_PIN1 PWR\_CSR\_EWUP**

## 35 HAL PWR Extension Driver

### 35.1 PWREx Firmware driver API description

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

#### 35.1.1 Peripheral extended features functions

##### Main and Backup Regulators configuration

- The backup domain includes 4 Kbytes of backup SRAM accessible only from the CPU, and address in 32-bit, 16-bit or 8-bit mode. Its content is retained even in Standby or VBAT mode when the low power backup regulator is enabled. It can be considered as an internal EEPROM when VBAT is always present. You can use the HAL\_PWR\_EnableBkUpReg() function to enable the low power backup regulator.
- When the backup domain is supplied by VDD (analog switch connected to VDD) the backup SRAM is powered from VDD which replaces the VBAT power supply to save battery life.
- The backup SRAM is not mass erased by a tamper event. It is read protected to prevent confidential data, such as cryptographic private key, from being accessed. The backup SRAM can be erased only through the Flash interface when a protection level change from level 1 to level 0 is requested. Refer to the description of Read protection (RDP) in the Flash programming manual.
- The main internal regulator can be configured to have a tradeoff between performance and power consumption when the device does not operate at the maximum frequency. This is done through \_\_HAL\_PWR\_MAINREGULATORMODE\_CONFIG() macro which configure VOS bit in PWR\_CR register Refer to the product datasheets for more details.

##### FLASH Power Down configuration

- By setting the FPDS bit in the PWR\_CR register by using the HAL\_PWR\_EnableFlashPowerDown() function, the Flash memory also enters power down mode when the device enters Stop mode. When the Flash memory is in power down mode, an additional startup delay is incurred when waking up from Stop mode.
- For STM32F42xxx/43xxx Devices, the scale can be modified only when the PLL is OFF and the HSI or HSE clock source is selected as system clock. The new value programmed is active only when the PLL is ON. When the PLL is OFF, the voltage scale 3 is automatically selected. Refer to the datasheets for more details.

##### Over-Drive and Under-Drive configuration

- For STM32F42xxx/43xxx Devices, in Run mode: the main regulator has 2 operating modes available:
  - Normal mode: The CPU and core logic operate at maximum frequency at a given voltage scaling (scale 1, scale 2 or scale 3)
  - Over-drive mode: This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale

2 or scale 3). This mode is enabled through HAL\_PWREx\_EnableOverDrive() function and disabled by HAL\_PWREx\_DisableOverDrive() function, to enter or exit from Over-drive mode please follow the sequence described in Reference manual.

- For STM32F42xxx/43xxx Devices, in Stop mode: the main regulator or low power regulator supplies a low power voltage to the 1.2V domain, thus preserving the content of registers and internal SRAM. 2 operating modes are available:
  - Normal mode: the 1.2V domain is preserved in nominal leakage mode. This mode is only available when the main regulator or the low power regulator is used in Scale 3 or low voltage mode.
  - Under-drive mode: the 1.2V domain is preserved in reduced leakage mode. This mode is only available when the main regulator or the low power regulator is in low voltage mode.
- [\*\*HAL\\_PWREx\\_EnableBkUpReg\(\)\*\*](#)
- [\*\*HAL\\_PWREx\\_DisableBkUpReg\(\)\*\*](#)
- [\*\*HAL\\_PWREx\\_EnableFlashPowerDown\(\)\*\*](#)
- [\*\*HAL\\_PWREx\\_DisableFlashPowerDown\(\)\*\*](#)
- [\*\*HAL\\_PWREx\\_ActivateOverDrive\(\)\*\*](#)
- [\*\*HAL\\_PWREx\\_DeactivateOverDrive\(\)\*\*](#)

### 35.1.2 Peripheral Extended features functions

#### 35.1.2.1 HAL\_PWREx\_EnableBkUpReg

Function Name	<b>HAL_StatusTypeDef HAL_PWREx_EnableBkUpReg ( void )</b>
Function Description	Enables the Backup Regulator.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 35.1.2.2 HAL\_PWREx\_DisableBkUpReg

Function Name	<b>HAL_StatusTypeDef HAL_PWREx_DisableBkUpReg ( void )</b>
Function Description	Disables the Backup Regulator.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 35.1.2.3 HAL\_PWREx\_EnableFlashPowerDown

Function Name	<b>void HAL_PWREx_EnableFlashPowerDown ( void )</b>
Function Description	Enables the Flash Power Down in Stop mode.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 35.1.2.4 HAL\_PWREx\_DisableFlashPowerDown

Function Name	<b>void HAL_PWREx_DisableFlashPowerDown ( void )</b>
Function Description	Disables the Flash Power Down in Stop mode.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 35.1.2.5 HAL\_PWREx\_ActivateOverDrive

Function Name	<b>HAL_StatusTypeDef HAL_PWREx_ActivateOverDrive ( void )</b>
Function Description	Activates the Over-Drive mode.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>These macros can be used only for STM32F42xx/STM32F43xx devices. This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale 2 or scale 3).</li><li>It is recommended to enter or exit Over-drive mode when the application is not running critical tasks and when the system clock source is either HSI or HSE. During the Over-drive</li></ul>

switch activation, no peripheral clocks should be enabled. The peripheral clocks must be enabled once the Over-drive mode is activated.

### 35.1.2.6 HAL\_PWREx\_DeactivateOverDrive

Function Name	<b>HAL_StatusTypeDef HAL_PWREx_DeactivateOverDrive ( void )</b>
Function Description	Deactivates the Over-Drive mode.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• These macros can be used only for STM32F42xx/STM32F43xx devices. This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale 2 or scale 3).</li> <li>• It is recommended to enter or exit Over-drive mode when the application is not running critical tasks and when the system clock source is either HSI or HSE. During the Over-drive switch activation, no peripheral clocks should be enabled. The peripheral clocks must be enabled once the Over-drive mode is activated.</li> </ul>

## 35.2 PWREx Firmware driver defines

### 35.2.1 PWREx

PWREx

*PWREx\_Over\_Under\_Drive\_Flag*

- #define: **PWR\_FLAG\_ODRDY PWR\_CSR\_ODRDY**
  
  
  
  
- #define: **PWR\_FLAG\_ODSWRDY PWR\_CSR\_ODSWRDY**
  
  
  
  
- #define: **PWR\_FLAG\_UDRDY PWR\_CSR\_UDSWRDY**



## 36 HAL RCC Generic Driver

### 36.1 RCC Firmware driver registers structures

#### 36.1.1 RCC\_PLLInitTypeDef

*RCC\_PLLInitTypeDef* is defined in the `stm32f4xx_hal_rcc.h`

##### Data Fields

- *uint32\_t PLLState*
- *uint32\_t PLLSource*
- *uint32\_t PLLM*
- *uint32\_t PLLN*
- *uint32\_t PLLP*
- *uint32\_t PLLQ*

##### Field Documentation

- *uint32\_t RCC\_PLLInitTypeDef::PLLState*
  - The new state of the PLL. This parameter can be a value of [\*RCC\\_PLL\\_Config\*](#)
- *uint32\_t RCC\_PLLInitTypeDef::PLLSource*
  - RCC\_PLLSource: PLL entry clock source. This parameter must be a value of [\*RCC\\_PLL\\_Clock\\_Source\*](#)
- *uint32\_t RCC\_PLLInitTypeDef::PLLM*
  - PLLM: Division factor for PLL VCO input clock. This parameter must be a number between Min\_Data = 0 and Max\_Data = 63
- *uint32\_t RCC\_PLLInitTypeDef::PLLN*
  - PLLN: Multiplication factor for PLL VCO output clock. This parameter must be a number between Min\_Data = 192 and Max\_Data = 432
- *uint32\_t RCC\_PLLInitTypeDef::PLLP*
  - PLLP: Division factor for main system clock (SYSCLK). This parameter must be a value of [\*RCC\\_PLLP\\_Clock\\_Divider\*](#)
- *uint32\_t RCC\_PLLInitTypeDef::PLLQ*
  - PLLQ: Division factor for OTG FS, SDIO and RNG clocks. This parameter must be a number between Min\_Data = 0 and Max\_Data = 63

#### 36.1.2 RCC\_ClkInitTypeDef

*RCC\_ClkInitTypeDef* is defined in the `stm32f4xx_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`](#)

### 36.1.3 `RCC_OscInitTypeDef`

`RCC_OscInitTypeDef` is defined in the `stm32f4xx_hal_rcc.h`

#### Data Fields

- **`uint32_t OscillatorType`**
- **`uint32_t HSEState`**
- **`uint32_t LSEState`**
- **`uint32_t HSISState`**
- **`uint32_t HSICalibrationValue`**
- **`uint32_t LSISState`**
- **`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::HSISState`**
  - 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::LSISState`**
  - The new state of the LSI. This parameter can be a value of [`RCC\_LSI\_Config`](#)

- ***RCC\_PLLInitTypeDef RCC\_OscInitTypeDef::PLL***
  - PLL structure parameters

## 36.2 RCC Firmware driver API description

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

### 36.2.1 RCC specific features

After reset the device is running from Internal High Speed oscillator (HSI 16MHz) with Flash 0 wait state, Flash prefetch buffer, D-Cache and I-Cache are 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 (I2S, RTC, ADC, USB OTG FS/SDIO/RNG)

### 36.2.2 Initialization and de-initialization functions

This section provides 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), 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
2. LSI (low-speed internal), 32 KHz low consumption RC used as IWDG and/or RTC clock source.
3. HSE (high-speed external), 4 to 26 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 two different output clocks:
  - The first output is used to generate the high speed system clock (up to 168 MHz)
  - The second output is used to generate the clock for the USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDIO (<= 48 MHz).
6. CSS (Clock security system), once enable using the macro `_HAL_RCC_CSS_ENABLE()` and if a HSE clock failure occurs(HSE used directly or through PLL as System clock source), the System clock is 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. MCO1 (microcontroller clock output), used to output HSI, LSE, HSE or PLL clock (through a configurable prescaler) on PA8 pin.
8. MCO2 (microcontroller clock output), used to output HSE, PLL, SYSCLK or PLLI2S clock (through a configurable prescaler) on PC9 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. All the peripheral clocks are derived from the System clock (SYSCLK) except: I2S: the I2S clock can be derived either from a specific PLL (PLLI2S) or from an external clock mapped on the I2S\_CKIN pin. You have to use \_\_HAL\_RCC\_PLLI2S\_CONFIG() macro to configure this clock. SAI: the SAI clock can be derived either from a specific PLL (PLLI2S) or (PLLSAI) or from an external clock mapped on the I2S\_CKIN pin. You have to use \_\_HAL\_RCC\_PLLI2S\_CONFIG() macro to configure this clock. RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use \_\_HAL\_RCC\_RTC\_CONFIG() and \_\_HAL\_RCC\_RTC\_ENABLE() macros to configure this clock. USB OTG FS, SDIO and RTC: USB OTG FS require a frequency equal to 48 MHz to work correctly, while the SDIO require a frequency equal or lower than to 48. This clock is derived of the main PLL through PLLQ divider. IWDG clock which is always the LSI clock.
2. For the STM32F405xx/07xx and STM32F415xx/17xx devices, the maximum frequency of the SYSCLK and HCLK is 168 MHz, PCLK2 84 MHz and PCLK1 42 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (refer to the product datasheets for more details).
3. For the STM32F42xxx and STM32F43xxx devices, the maximum frequency of the SYSCLK and HCLK is 180 MHz, PCLK2 90 MHz and PCLK1 45 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (refer to the product datasheets for more details).
4. For the STM32F401xx, the maximum frequency of the SYSCLK and HCLK is 84 MHz, PCLK2 84 MHz and PCLK1 42 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (refer to the product datasheets for more details).
  - [\*\*HAL\\_RCC\\_DelInit\(\)\*\*](#)
  - [\*\*HAL\\_RCC\\_OscConfig\(\)\*\*](#)
  - [\*\*HAL\\_RCC\\_ClockConfig\(\)\*\*](#)

### 36.2.3 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\(\)\*](#)

### 36.2.4 Initialization and de-initialization functions

#### 36.2.4.1 HAL\_RCC\_DeInit

Function Name	<b>void HAL_RCC_DeInit ( void )</b>
Function Description	Resets the RCC clock configuration to the default reset state.
Parameters	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, PLL and PLLI2S OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO1 and MCO2 OFFAll interrupts disabled</li> <li>• This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks</li> </ul>

#### 36.2.4.2 HAL\_RCC\_OscConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCC_OscConfig (</b> <b><i>RCC_OscInitTypeDef * RCC_OscInitStruct</i></b> )
Function Description	Initializes the RCC Oscillators according to the specified parameters in the <b>RCC_OscInitTypeDef</b> .
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_OscInitStruct</b> : pointer to an <b>RCC_OscInitTypeDef</b> structure that contains the configuration information for the RCC Oscillators.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The PLL is not disabled when used as system clock.</li> </ul>

#### 36.2.4.3 HAL\_RCC\_ClockConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCC_ClockConfig (</b> <b>RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)</b>
Function Description	Initializes the CPU, AHB and APB busses clocks according to the specified parameters in the <b>RCC_ClkInitStruct</b> .
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_ClkInitStruct</b> : pointer to an <b>RCC_OscInitTypeDef</b> structure that contains the configuration information for the RCC peripheral.</li> <li>• <b>FLatency</b> : FLASH Latency, this parameter depend on device selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by <b>HAL_RCC_GetHCLKFreq()</b> function called within this function</li> <li>• 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).</li> <li>• 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.</li> <li>• Depending on the device voltage range, the software has to set correctly HPRE[3:0] bits to ensure that HCLK not exceed the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")</li> </ul>

### 36.2.5 Peripheral Control functions

#### 36.2.5.1 HAL\_RCC\_MCOConfig

Function Name	<b>void HAL_RCC_MCOConfig ( uint32_t RCC_MCOx, uint32_t RCC_MCOsource, uint32_t RCC_MCODiv)</b>
Function Description	Selects the clock source to output on MCO1 pin(PA8) or on MCO2 pin(PC9).
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_MCOx</b> : specifies the output direction for the clock source. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RCC_MCO1</b> : Clock source to output on MCO1 pin(PA8).</li> <li>– <b>RCC_MCO2</b> : Clock source to output on MCO2 pin(PC9).</li> </ul> </li> <li>• <b>RCC_MCOsource</b> : specifies the clock source to output. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RCC_MCO1SOURCE_HSI</b> : HSI clock selected as</li> </ul> </li> </ul>

	MCO1 source
	– <b>RCC_MCO1SOURCE_LSE</b> : LSE clock selected as MCO1 source
	– <b>RCC_MCO1SOURCE_HSE</b> : HSE clock selected as MCO1 source
	– <b>RCC_MCO1SOURCE_PLLCLK</b> : main PLL clock selected as MCO1 source
	– <b>RCC_MCO2SOURCE_SYSCLK</b> : System clock (SYSCLK) selected as MCO2 source
	– <b>RCC_MCO2SOURCE_PLLI2SCLK</b> : PLLI2S clock selected as MCO2 source
	– <b>RCC_MCO2SOURCE_HSE</b> : HSE clock selected as MCO2 source
	– <b>RCC_MCO2SOURCE_PLLCLK</b> : main PLL clock selected as MCO2 source
•	<b>RCC_MCODiv</b> : specifies the MCOx prescaler. This parameter can be one of the following values:
	– <b>RCC_MCODIV_1</b> : no division applied to MCOx clock
	– <b>RCC_MCODIV_2</b> : division by 2 applied to MCOx clock
	– <b>RCC_MCODIV_3</b> : division by 3 applied to MCOx clock
	– <b>RCC_MCODIV_4</b> : division by 4 applied to MCOx clock
	– <b>RCC_MCODIV_5</b> : division by 5 applied to MCOx clock
Return values	• None.
Notes	• PA8/PC9 should be configured in alternate function mode.

### 36.2.5.2 HAL\_RCC\_EnableCSS

Function Name	<b>void HAL_RCC_EnableCSS ( void )</b>
Function Description	Enables the Clock Security System.
Parameters	• None.
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.

### 36.2.5.3 HAL\_RCC\_DisableCSS

Function Name	<b>void HAL_RCC_DisableCSS ( void )</b>
Function Description	Disables the Clock Security System.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

### 36.2.5.4 HAL\_RCC\_GetSysClockFreq

Function Name	<b>uint32_t HAL_RCC_GetSysClockFreq ( void )</b>
Function Description	Returns the SYSCLK frequency.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>SYSCLK frequency</b></li></ul>
Notes	<ul style="list-style-type: none"><li>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:<ul style="list-style-type: none"><li>If SYSCLK source is HSI, function returns values based on HSI_VALUE(*)</li><li>If SYSCLK source is HSE, function returns values based on HSE_VALUE(**)</li><li>If SYSCLK source is PLL, function returns values based on HSE_VALUE(**) or HSI_VALUE(*) multiplied/divided by the PLL factors.</li></ul></li><li>(*) HSI_VALUE is a constant defined in stm32f4xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.</li><li>(**) HSE_VALUE is a constant defined in stm32f4xx_hal_conf.h file (default value 25 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.</li><li>The result of this function could be not correct when using fractional value for HSE crystal.</li><li>This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.</li><li>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.</li></ul>

### 36.2.5.5 HAL\_RCC\_GetHCLKFreq

Function Name	<b>uint32_t HAL_RCC_GetHCLKFreq ( void )</b>
Function Description	Returns the HCLK frequency.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>HCLK frequency</b></li></ul>
Notes	<ul style="list-style-type: none"><li>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.</li><li>The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function</li></ul>

### 36.2.5.6 HAL\_RCC\_GetPCLK1Freq

Function Name	<b>uint32_t HAL_RCC_GetPCLK1Freq ( void )</b>
Function Description	Returns the PCLK1 frequency.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>PCLK1 frequency</b></li></ul>
Notes	<ul style="list-style-type: none"><li>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.</li></ul>

### 36.2.5.7 HAL\_RCC\_GetPCLK2Freq

Function Name	<b>uint32_t HAL_RCC_GetPCLK2Freq ( void )</b>
Function Description	Returns the PCLK2 frequency.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li><b>PCLK2 frequency</b></li></ul>

## 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.

### 36.2.5.8 HAL\_RCC\_GetOscConfig

Function Name	<code>void HAL_RCC_GetOscConfig ( <i>RCC_OscInitTypeDef</i> * RCC_OscInitStruct)</code>
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"><li>• <b>RCC_OscInitStruct</b> : pointer to an RCC_OscInitTypeDef structure that will be configured.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 36.2.5.9 HAL\_RCC\_GetClockConfig

Function Name	<code>void HAL_RCC_GetClockConfig ( <i>RCC_ClkInitTypeDef</i> * RCC_ClkInitStruct, uint32_t * pFLatency)</code>
Function Description	Configures the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"><li>• <b>RCC_OscInitStruct</b> : pointer to an RCC_OscInitTypeDef structure that will be configured.</li><li>• <b>pFLatency</b> : Pointer on the Flash Latency.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 36.2.5.10 HAL\_RCC\_NMI\_IRQHandler

Function Name	<b>void HAL_RCC_NMI_IRQHandler ( void )</b>
Function Description	This function handles the RCC CSS interrupt request.
Parameters	<ul style="list-style-type: none"><li>None.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>This API should be called under the NMI_Handler().</li></ul>

### 36.2.5.11 HAL\_RCC\_CCSCallback

Function Name	<b>void HAL_RCC_CCSCallback ( void )</b>
Function Description	RCC Clock Security System interrupt callback.
Parameters	<ul style="list-style-type: none"><li><b>none</b> :</li></ul>
Return values	<ul style="list-style-type: none"><li><b>none</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

## 36.3 RCC Firmware driver defines

### 36.3.1 RCC

RCC

*RCC\_AHB\_Clock\_Source*

- #define: **RCC\_SYSCLK\_DIV1 RCC\_CFGR\_HPRE\_DIV1**
- #define: **RCC\_SYSCLK\_DIV2 RCC\_CFGR\_HPRE\_DIV2**
- #define: **RCC\_SYSCLK\_DIV4 RCC\_CFGR\_HPRE\_DIV4**
- #define: **RCC\_SYSCLK\_DIV8 RCC\_CFGR\_HPRE\_DIV8**

- #define: **RCC\_SYSCLK\_DIV16 RCC\_CFGR\_HPRE\_DIV16**
- #define: **RCC\_SYSCLK\_DIV64 RCC\_CFGR\_HPRE\_DIV64**
- #define: **RCC\_SYSCLK\_DIV128 RCC\_CFGR\_HPRE\_DIV128**
- #define: **RCC\_SYSCLK\_DIV256 RCC\_CFGR\_HPRE\_DIV256**
- #define: **RCC\_SYSCLK\_DIV512 RCC\_CFGR\_HPRE\_DIV512**

#### ***RCC\_APB1\_APB2\_Clock\_Source***

- #define: **RCC\_HCLK\_DIV1 RCC\_CFGR\_PPREG1\_DIV1**
- #define: **RCC\_HCLK\_DIV2 RCC\_CFGR\_PPREG1\_DIV2**
- #define: **RCC\_HCLK\_DIV4 RCC\_CFGR\_PPREG1\_DIV4**
- #define: **RCC\_HCLK\_DIV8 RCC\_CFGR\_PPREG1\_DIV8**
- #define: **RCC\_HCLK\_DIV16 RCC\_CFGR\_PPREG1\_DIV16**

#### ***RCC\_BitAddress\_AliasRegion***

- #define: **RCC\_OFFSET (RCC\_BASE - PERIPH\_BASE)**
- #define: **RCC\_CR\_OFFSET (RCC\_OFFSET + 0x00)**

- #define: ***HSION\_BitNumber 0x00***
  
- #define: ***CR\_HSION\_BB (PERIPH\_BB\_BASE + (RCC\_CR\_OFFSET \* 32) + (HSION\_BitNumber \* 4))***
  
- #define: ***CSSON\_BitNumber 0x13***
  
- #define: ***CR\_CSSON\_BB (PERIPH\_BB\_BASE + (RCC\_CR\_OFFSET \* 32) + (CSSON\_BitNumber \* 4))***
  
- #define: ***PLLON\_BitNumber 0x18***
  
- #define: ***CR\_PLLON\_BB (PERIPH\_BB\_BASE + (RCC\_CR\_OFFSET \* 32) + (PLLON\_BitNumber \* 4))***
  
- #define: ***PLL12SON\_BitNumber 0x1A***
  
- #define: ***CR\_PLLI2SON\_BB (PERIPH\_BB\_BASE + (RCC\_CR\_OFFSET \* 32) + (PLL12SON\_BitNumber \* 4))***
  
- #define: ***RCC\_CFGR\_OFFSET (RCC\_OFFSET + 0x08)***
  
- #define: ***I2SSRC\_BitNumber 0x17***
  
- #define: ***CFGGR\_I2SSRC\_BB (PERIPH\_BB\_BASE + (RCC\_CFGR\_OFFSET \* 32) + (I2SSRC\_BitNumber \* 4))***

- #define: **RCC\_BDCR\_OFFSET** (**RCC\_OFFSET** + 0x70)
- #define: **RTCEN\_BitNumber** 0x0F
- #define: **BDCR\_RTCEN\_BB** (**PERIPH\_BB\_BASE** + (**RCC\_BDCR\_OFFSET** \* 32) + (**RTCEN\_BitNumber** \* 4))
- #define: **BDRST\_BitNumber** 0x10
- #define: **BDCR\_BDRST\_BB** (**PERIPH\_BB\_BASE** + (**RCC\_BDCR\_OFFSET** \* 32) + (**BDRST\_BitNumber** \* 4))
- #define: **RCC\_CSR\_OFFSET** (**RCC\_OFFSET** + 0x74)
- #define: **LSION\_BitNumber** 0x00
- #define: **CSR\_LSION\_BB** (**PERIPH\_BB\_BASE** + (**RCC\_CSR\_OFFSET** \* 32) + (**LSION\_BitNumber** \* 4))
- #define: **CR\_BYTE2\_ADDRESS** ((**uint32\_t**)0x40023802)
- #define: **CIR\_BYTE1\_ADDRESS** ((**uint32\_t**)(**RCC\_BASE** + 0x0C + 0x01))
- #define: **CIR\_BYTE2\_ADDRESS** ((**uint32\_t**)(**RCC\_BASE** + 0x0C + 0x02))

- #define: **BDCR\_BYTE0\_ADDRESS (PERIPH\_BASE + RCC\_BDCR\_OFFSET)**
- #define: **DBP\_TIMEOUT\_VALUE ((uint32\_t)100)**
- #define: **LSE\_TIMEOUT\_VALUE ((uint32\_t)5000)**

#### **RCC\_Flag**

- #define: **RCC\_FLAG\_HSIRDY ((uint8\_t)0x21)**
- #define: **RCC\_FLAG\_HSERDY ((uint8\_t)0x31)**
- #define: **RCC\_FLAG\_PLLRDY ((uint8\_t)0x39)**
- #define: **RCC\_FLAG\_PLLI2SRDY ((uint8\_t)0x3B)**
- #define: **RCC\_FLAG\_LSERDY ((uint8\_t)0x41)**
- #define: **RCC\_FLAG\_LSIRDY ((uint8\_t)0x61)**
- #define: **RCC\_FLAG\_BORRST ((uint8\_t)0x79)**
- #define: **RCC\_FLAG\_PINRST ((uint8\_t)0x7A)**
- #define: **RCC\_FLAG\_PORRST ((uint8\_t)0x7B)**

- #define: **RCC\_FLAG\_SFTRST** ((*uint8\_t*)0x7C)
- #define: **RCC\_FLAG\_IWDGRST** ((*uint8\_t*)0x7D)
- #define: **RCC\_FLAG\_WWDGRST** ((*uint8\_t*)0x7E)
- #define: **RCC\_FLAG\_LPWRREST** ((*uint8\_t*)0x7F)

#### ***RCC\_HSE\_Config***

- #define: **RCC\_HSE\_OFF** ((*uint8\_t*)0x00)
- #define: **RCC\_HSE\_ON** ((*uint8\_t*)0x01)
- #define: **RCC\_HSE\_BYPASS** ((*uint8\_t*)0x05)

#### ***RCC\_HSI\_Config***

- #define: **RCC\_HSI\_OFF** ((*uint8\_t*)0x00)
- #define: **RCC\_HSI\_ON** ((*uint8\_t*)0x01)

#### ***RCC\_I2S\_Clock\_Source***

- #define: **RCC\_I2SCLKSOURCE\_PLLI2S** ((*uint32\_t*)0x00000000)
- #define: **RCC\_I2SCLKSOURCE\_EXT** ((*uint32\_t*)0x00000001)

***RCC\_Interrupt***

- #define: ***RCC\_IT\_LSIRDY*** ((*uint8\_t*)0x01)
- #define: ***RCC\_IT\_LSERDY*** ((*uint8\_t*)0x02)
- #define: ***RCC\_IT\_HSIRDY*** ((*uint8\_t*)0x04)
- #define: ***RCC\_IT\_HSERDY*** ((*uint8\_t*)0x08)
- #define: ***RCC\_IT\_PLLRDY*** ((*uint8\_t*)0x10)
- #define: ***RCC\_IT\_PLLI2SRDY*** ((*uint8\_t*)0x20)
- #define: ***RCC\_IT\_CSS*** ((*uint8\_t*)0x80)

***RCC\_LSE\_Config***

- #define: ***RCC\_LSE\_OFF*** ((*uint8\_t*)0x00)
- #define: ***RCC\_LSE\_ON*** ((*uint8\_t*)0x01)
- #define: ***RCC\_LSE\_BYPASS*** ((*uint8\_t*)0x05)

***RCC\_LSI\_Config***

- #define: ***RCC\_LSI\_OFF*** ((*uint8\_t*)0x00)

- #define: **RCC\_LSI\_ON** ((*uint8\_t*)0x01)

#### ***RCC\_MCO1\_Clock\_Source***

- #define: **RCC\_MCO1SOURCE\_HSI** ((*uint32\_t*)0x00000000)
- #define: **RCC\_MCO1SOURCE\_LSE** **RCC\_CFGR\_MCO1\_0**
- #define: **RCC\_MCO1SOURCE\_HSE** **RCC\_CFGR\_MCO1\_1**
- #define: **RCC\_MCO1SOURCE\_PLLCLK** **RCC\_CFGR\_MCO1**

#### ***RCC\_MCO2\_Clock\_Source***

- #define: **RCC\_MCO2SOURCE\_SYSCLK** ((*uint32\_t*)0x00000000)
- #define: **RCC\_MCO2SOURCE\_PLLI2SCLK** **RCC\_CFGR\_MCO2\_0**
- #define: **RCC\_MCO2SOURCE\_HSE** **RCC\_CFGR\_MCO2\_1**
- #define: **RCC\_MCO2SOURCE\_PLLCLK** **RCC\_CFGR\_MCO2**

#### ***RCC\_MCOx\_Clock\_Prescaler***

- #define: **RCC\_MCODIV\_1** ((*uint32\_t*)0x00000000)
- #define: **RCC\_MCODIV\_2** **RCC\_CFGR\_MCO1PRE\_2**

- #define: **RCC\_MCODIV\_3** ((*uint32\_t*)RCC\_CFGR\_MCO1PRE\_0 |  
  RCC\_CFGR\_MCO1PRE\_2)
- #define: **RCC\_MCODIV\_4** ((*uint32\_t*)RCC\_CFGR\_MCO1PRE\_1 |  
  RCC\_CFGR\_MCO1PRE\_2)
- #define: **RCC\_MCODIV\_5** RCC\_CFGR\_MCO1PRE

#### **RCC\_MCO\_Index**

- #define: **RCC\_MCO1** ((*uint32\_t*)0x00000000)
- #define: **RCC\_MCO2** ((*uint32\_t*)0x00000001)

#### **RCC\_Oscillator\_Type**

- #define: **RCC\_OSCILLATORTYPE\_NONE** ((*uint32\_t*)0x00000000)
- #define: **RCC\_OSCILLATORTYPE\_HSE** ((*uint32\_t*)0x00000001)
- #define: **RCC\_OSCILLATORTYPE\_HSI** ((*uint32\_t*)0x00000002)
- #define: **RCC\_OSCILLATORTYPE\_LSE** ((*uint32\_t*)0x00000004)
- #define: **RCC\_OSCILLATORTYPE\_LSI** ((*uint32\_t*)0x00000008)

#### **RCC\_PLLP\_Clock\_Divider**

- #define: **RCC\_PLLP\_DIV2** ((*uint32\_t*)0x00000002)
- #define: **RCC\_PLLP\_DIV4** ((*uint32\_t*)0x00000004)
- #define: **RCC\_PLLP\_DIV6** ((*uint32\_t*)0x00000006)
- #define: **RCC\_PLLP\_DIV8** ((*uint32\_t*)0x00000008)

#### **RCC\_PLL\_Clock\_Source**

- #define: **RCC\_PLLSOURCE\_HSI** **RCC\_PLLCFGR\_PLLSRC\_HSI**
- #define: **RCC\_PLLSOURCE\_HSE** **RCC\_PLLCFGR\_PLLSRC\_HSE**

#### **RCC\_PLL\_Config**

- #define: **RCC\_PLL\_NONE** ((*uint8\_t*)0x00)
- #define: **RCC\_PLL\_OFF** ((*uint8\_t*)0x01)
- #define: **RCC\_PLL\_ON** ((*uint8\_t*)0x02)

#### **RCC\_RTC\_Clock\_Source**

- #define: **RCC\_RTCCLKSOURCE\_LSE** ((*uint32\_t*)0x00000100)
- #define: **RCC\_RTCCLKSOURCE\_LSI** ((*uint32\_t*)0x00000200)

- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV2** ((*uint32\_t*)0x00020300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV3** ((*uint32\_t*)0x00030300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV4** ((*uint32\_t*)0x00040300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV5** ((*uint32\_t*)0x00050300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV6** ((*uint32\_t*)0x00060300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV7** ((*uint32\_t*)0x00070300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV8** ((*uint32\_t*)0x00080300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV9** ((*uint32\_t*)0x00090300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV10** ((*uint32\_t*)0x000A0300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV11** ((*uint32\_t*)0x000B0300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV12** ((*uint32\_t*)0x000C0300)
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV13** ((*uint32\_t*)0x000D0300)

- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV14 ((uint32\_t)0x000E0300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV15 ((uint32\_t)0x000F0300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV16 ((uint32\_t)0x00100300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV17 ((uint32\_t)0x00110300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV18 ((uint32\_t)0x00120300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV19 ((uint32\_t)0x00130300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV20 ((uint32\_t)0x00140300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV21 ((uint32\_t)0x00150300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV22 ((uint32\_t)0x00160300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV23 ((uint32\_t)0x00170300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV24 ((uint32\_t)0x00180300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV25 ((uint32\_t)0x00190300)**

- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV26 ((uint32\_t)0x001A0300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV27 ((uint32\_t)0x001B0300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV28 ((uint32\_t)0x001C0300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV29 ((uint32\_t)0x001D0300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV30 ((uint32\_t)0x001E0300)**
- #define: **RCC\_RTCCLKSOURCE\_HSE\_DIV31 ((uint32\_t)0x001F0300)**

#### ***RCC\_System\_Clock\_Source***

- #define: **RCC\_SYSCLKSOURCE\_HSI RCC\_CFGR\_SW\_HSI**
- #define: **RCC\_SYSCLKSOURCE\_HSE RCC\_CFGR\_SW\_HSE**
- #define: **RCC\_SYSCLKSOURCE\_PLLCLK RCC\_CFGR\_SW\_PLL**

#### ***RCC\_System\_Clock\_Type***

- #define: **RCC\_CLOCKTYPE\_SYSCLK ((uint32\_t)0x00000001)**
- #define: **RCC\_CLOCKTYPE\_HCLK ((uint32\_t)0x00000002)**
- #define: **RCC\_CLOCKTYPE\_PCLK1 ((uint32\_t)0x00000004)**

- #define: **RCC\_CLOCKTYPE\_PCLK2 ((uint32\_t)0x00000008)**

## 37 HAL RCC Extension Driver

### 37.1 RCCEx Firmware driver registers structures

#### 37.1.1 RCC\_PLLI2SInitTypeDef

*RCC\_PLLI2SInitTypeDef* is defined in the `stm32f4xx_hal_rcc_ex.h`

##### Data Fields

- *uint32\_t PLLI2SN*
- *uint32\_t PLLI2SR*
- *uint32\_t PLLI2SQ*

##### Field Documentation

- *uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SN*
  - Specifies the multiplication factor for PLLI2S VCO output clock. This parameter must be a number between Min\_Data = 192 and Max\_Data = 432. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI
- *uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SR*
  - Specifies the division factor for I2S clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 7. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI
- *uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SQ*
  - Specifies the division factor for SAI1 clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 15. This parameter will be used only when PLLI2S is selected as Clock Source SAI

#### 37.1.2 RCC\_PLLSAInitTypeDef

*RCC\_PLLSAInitTypeDef* is defined in the `stm32f4xx_hal_rcc_ex.h`

##### Data Fields

- *uint32\_t PLLSAIN*
- *uint32\_t PLLSAIQ*
- *uint32\_t PLLSAIR*

##### Field Documentation

- *uint32\_t RCC\_PLLSAInitTypeDef::PLLSAIN*

- Specifies the multiplication factor for PLLI2S VCO output clock. This parameter must be a number between Min\_Data = 192 and Max\_Data = 432. This parameter will be used only when PLLSAI is selected as Clock Source SAI or LTDC
- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIQ***
  - Specifies the division factor for SAI1 clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 15. This parameter will be used only when PLLSAI is selected as Clock Source SAI or LTDC
- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIR***
  - specifies the division factor for LTDC clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 7. This parameter will be used only when PLLSAI is selected as Clock Source LTDC

### 37.1.3 RCC\_PeriphCLKInitTypeDef

*RCC\_PeriphCLKInitTypeDef* is defined in the `stm32f4xx_hal_rcc_ex.h`

#### Data Fields

- ***uint32\_t PeriphClockSelection***
- ***RCC\_PLLI2SInitTypeDef PLLI2S***
- ***RCC\_PLLSAIInitTypeDef PLLSAI***
- ***uint32\_t PLLI2SDivQ***
- ***uint32\_t PLLSAIDivQ***
- ***uint32\_t PLLSAIDivR***
- ***uint32\_t RTCClockSelection***
- ***uint8\_t TIMPresSelection***

#### Field Documentation

- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PeriphClockSelection***
  - The Extended Clock to be configured. This parameter can be a value of [\*RCCEx\\_Periph\\_Clock\\_Selection\*](#)
- ***RCC\_PLLI2SInitTypeDef RCC\_PeriphCLKInitTypeDef::PLLI2S***
  - PLL I2S structure parameters. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI
- ***RCC\_PLLSAIInitTypeDef RCC\_PeriphCLKInitTypeDef::PLLSAI***
  - PLL SAI structure parameters. This parameter will be used only when PLLI2S is selected as Clock Source SAI or LTDC
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PLLI2SDivQ***
  - Specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between Min\_Data = 1 and Max\_Data = 32. This parameter will be used only when PLLI2S is selected as Clock Source SAI
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PLLSAIDivQ***
  - Specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between Min\_Data = 1 and Max\_Data = 32. This parameter will be used only when PLLSAI is selected as Clock Source SAI
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PLLSAIDivR***
  - Specifies the PLLSAI division factor for LTDC clock. This parameter must be one value of [\*RCCEx\\_PLLSAI\\_DIVR\*](#)

- ***uint32\_t RCC\_PeriphCLKInitTypeDef::RTCClockSelection***
  - Specifies RTC Clock Prescalers Selection This parameter can be a value of **RCC\_RTC\_Clock\_Source**
- ***uint8\_t RCC\_PeriphCLKInitTypeDef::TIMPresSelection***
  - Specifies TIM Clock Prescalers Selection This parameter can be a value of **RCCEEx\_TIM\_Prescaler\_Selection**

## 37.2 RCCEEx Firmware driver API description

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

### 37.2.1 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\_RCCEEx\_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC\_BDCR register are set to their reset values.

- ***HAL\_RCCEEx\_PeriphCLKConfig()***
- ***HAL\_RCCEEx\_GetPeriphCLKConfig()***

### 37.2.2 Extended Peripheral Control functions

#### 37.2.2.1 HAL\_RCCEEx\_PeriphCLKConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCCEEx_PeriphCLKConfig (</b> <b><i>RCC_PeriphCLKInitTypeDef * PeriphClkInit</i></b> )
Function Description	Initializes the RCC extended peripherals clocks according to the specified parameters in the <b>RCC_PeriphCLKInitTypeDef</b> .
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit</b> : pointer to an <b>RCC_PeriphCLKInitTypeDef</b> structure that contains the configuration information for the Extended Peripherals clocks(I2S and RTC clocks).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit</b> : pointer to an <b>RCC_PeriphCLKInitTypeDef</b> structure that contains the configuration information for the Extended Peripherals clocks(I2S, SAI, LTDC RTC and TIM).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit</b> : pointer to an <b>RCC_PeriphCLKInitTypeDef</b> structure that contains the configuration information for the Extended Peripherals clocks(I2S and RTC clocks).</li> </ul>

Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>A caution to be taken when HAL_RCCEEx_PeriphCLKConfig() is used to select RTC clock selection, in this case the Reset of Backup domain will be applied in order to modify the RTC Clock source as consequence all backup domain (RTC and RCC_BDCR register expect BKPSRAM) will be reset</li> <li>Care must be taken when HAL_RCCEEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.</li> <li>A caution to be taken when HAL_RCCEEx_PeriphCLKConfig() is used to select RTC clock selection, in this case the Reset of Backup domain will be applied in order to modify the RTC Clock source as consequence all backup domain (RTC and RCC_BDCR register expect BKPSRAM) will be reset</li> </ul>

### 37.2.2.2 HAL\_RCCEEx\_GetPeriphCLKConfig

Function Name	<code>void HAL_RCCEEx_GetPeriphCLKConfig (</code> <i>RCC_PeriphCLKInitTypeDef * PeriphClkInit)</i>
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li><b>PeriphClkInit</b> : pointer to an RCC_PeriphCLKInitTypeDef structure that will be configured.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 37.3 RCCEx Firmware driver defines

### 37.3.1 RCCEx

RCCEx

*RCCEx\_BitAddress\_AliasRegion*

- #define: **PLLSAION\_BitNumber 0x1C**

- #define: ***CR\_PLLSAION\_BB (PERIPH\_BB\_BASE + (RCC\_CR\_OFFSET \* 32) + (PLL\_SAION\_BitNumber \* 4))***
- #define: ***RCC\_DCKCFGR\_OFFSET (RCC\_OFFSET + 0x8C)***
- #define: ***TIMPRE\_BitNumber 0x18***
- #define: ***DCKCFGR\_TIMPRE\_BB (PERIPH\_BB\_BASE + (RCC\_DCKCFGR\_OFFSET \* 32) + (TIMPRE\_BitNumber \* 4))***

#### ***RCCEEx\_Periph\_Clock\_Selection***

- #define: ***RCC\_PERIPHCLK\_I2S ((uint32\_t)0x00000001)***
- #define: ***RCC\_PERIPHCLK\_SAI\_PLLI2S ((uint32\_t)0x00000002)***
- #define: ***RCC\_PERIPHCLK\_SAI\_PLLSAI ((uint32\_t)0x00000004)***
- #define: ***RCC\_PERIPHCLK\_LTDC ((uint32\_t)0x00000008)***
- #define: ***RCC\_PERIPHCLK\_TIM ((uint32\_t)0x00000010)***
- #define: ***RCC\_PERIPHCLK\_RTC ((uint32\_t)0x00000020)***

#### ***RCCEEx\_PLLSAI\_DIVR***

- #define: ***RCC\_PLLSAIDIVR\_2 ((uint32\_t)0x00000000)***

- #define: **RCC\_PLLSAIDIVR\_4 ((uint32\_t)0x00010000)**

- #define: **RCC\_PLLSAIDIVR\_8 ((uint32\_t)0x00020000)**

- #define: **RCC\_PLLSAIDIVR\_16 ((uint32\_t)0x00030000)**

#### ***RCCEEx\_SAI\_BlockA\_Clock\_Source***

- #define: **RCC\_SAIACLKSOURCE\_PLLSAI ((uint32\_t)0x00000000)**

- #define: **RCC\_SAIACLKSOURCE\_PLLI2S ((uint32\_t)0x00100000)**

- #define: **RCC\_SAIACLKSOURCE\_EXT ((uint32\_t)0x00200000)**

#### ***RCCEEx\_SAI\_BlockB\_Clock\_Source***

- #define: **RCC\_SAIBCLKSOURCE\_PLLSAI ((uint32\_t)0x00000000)**

- #define: **RCC\_SAIBCLKSOURCE\_PLLI2S ((uint32\_t)0x00400000)**

- #define: **RCC\_SAIBCLKSOURCE\_EXT ((uint32\_t)0x00800000)**

#### ***RCCEEx\_TIM\_PRescaler\_Selection***

- #define: **RCC\_TIMPRES\_DESACTIVATED ((uint8\_t)0x00)**

- #define: **RCC\_TIMPRES\_ACTIVATED ((uint8\_t)0x01)**

## 38 HAL RNG Generic Driver

### 38.1 RNG Firmware driver registers structures

#### 38.1.1 RNG\_HandleTypeDef

*RNG\_HandleTypeDef* is defined in the `stm32f4xx_hal_rng.h`

##### Data Fields

- *RNG\_TypeDef \* Instance*  
– Register base address
- *HAL\_LockTypeDef Lock*  
– RNG locking object
- *\_\_IO HAL\_RNG\_StateTypeDef State*  
– RNG communication state

##### Field Documentation

- *RNG\_TypeDef\* RNG\_HandleTypeDef::Instance*  
– Register base address
- *HAL\_LockTypeDef RNG\_HandleTypeDef::Lock*  
– RNG locking object
- *\_\_IO HAL\_RNG\_StateTypeDef RNG\_HandleTypeDef::State*  
– RNG communication state

#### 38.1.2 RNG\_TypeDef

*RNG\_TypeDef* is defined in the `stm32f439xx.h`

##### Data Fields

- *\_\_IO uint32\_t CR*
- *\_\_IO uint32\_t SR*
- *\_\_IO uint32\_t DR*

##### Field Documentation

- *\_\_IO uint32\_t RNG\_TypeDef::CR*  
– RNG control register, Address offset: 0x00
- *\_\_IO uint32\_t RNG\_TypeDef::SR*  
– RNG status register, Address offset: 0x04
- *\_\_IO uint32\_t RNG\_TypeDef::DR*  
– RNG data register, Address offset: 0x08

## 38.2 RNG Firmware driver API description

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

### 38.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using \_\_RNG\_CLK\_ENABLE() macro.
2. Activate the RNG peripheral using \_\_HAL\_RNG\_ENABLE() macro.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using HAL\_RNG\_GetRandomNumber() function.

### 38.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the RNG\_InitTypeDef and create the associated handle
- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP
- [\*\*HAL\\_RNG\\_Init\(\)\*\*](#)
- [\*\*HAL\\_RNG\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_RNG\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_RNG\\_MspDeInit\(\)\*\*](#)

### 38.2.3 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- handle RNG interrupt request
- [\*\*HAL\\_RNG\\_GetRandomNumber\(\)\*\*](#)
- [\*\*HAL\\_RNG\\_GetRandomNumber\\_IT\(\)\*\*](#)
- [\*\*HAL\\_RNG\\_IRQHandler\(\)\*\*](#)
- [\*\*HAL\\_RNG\\_ReadyCallback\(\)\*\*](#)
- [\*\*HAL\\_RNG\\_ErrorCallback\(\)\*\*](#)

### 38.2.4 Peripheral State functions

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

- [\*\*HAL\\_RNG\\_GetState\(\)\*\*](#)

### 38.2.5 Initialization and de-initialization functions

#### 38.2.5.1 HAL\_RNG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_RNG_Init ( <i>RNG_HandleTypeDef</i> * <i>hrng</i>)</b>
Function Description	Initializes the RNG according to the specified parameters in the <i>RNG_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b> : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.5.2 HAL\_RNG\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_RNG_DelInit ( <i>RNG_HandleTypeDef</i> * <i>hrng</i>)</b>
Function Description	Deinitializes the RNG peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b> : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.5.3 HAL\_RNG\_MsplInit

Function Name	<b>void HAL_RNG_MsplInit ( <i>RNG_HandleTypeDef</i> * <i>hrng</i>)</b>
Function Description	Initializes the RNG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b> : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**38.2.5.4 HAL\_RNG\_MspDeInit**

Function Name	<b>void HAL_RNG_MspDeInit ( <i>RNG_HandleTypeDef</i> * hrng)</b>
Function Description	Deinitializes the RNG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b> : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**38.2.6 Peripheral Control functions****38.2.6.1 HAL\_RNG\_GetRandomNumber**

Function Name	<b>uint32_t HAL_RNG_GetRandomNumber ( <i>RNG_HandleTypeDef</i> * hrng)</b>
Function Description	Returns a 32-bit random number.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b> : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>32-bit random number</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Each time the random number data is read the <i>RNG_FLAG_DRDY</i> flag is automatically cleared.</li> </ul>

**38.2.6.2 HAL\_RNG\_GetRandomNumber\_IT**

Function Name	<b>uint32_t HAL_RNG_GetRandomNumber_IT ( <i>RNG_HandleTypeDef</i> * hrng)</b>
Function Description	Returns a 32-bit random number with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b> : pointer to a <i>RNG_HandleTypeDef</i> structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>32-bit random number</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.6.3 HAL\_RNG\_IRQHandler

Function Name	<b>void HAL_RNG_IRQHandler ( <i>RNG_HandleTypeDef</i> * hrng)</b>
Function Description	Handles RNG interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b> : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using <code>_HAL_RNG_CLEAR_FLAG()</code>. The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used.</li> <li>• In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using <code>_HAL_RNG_CLEAR_FLAG()</code>, then disable and enable the RNG peripheral to reinitialize and restart the RNG.</li> </ul>

### 38.2.6.4 HAL\_RNG\_ReadyCallback

Function Name	<b>void HAL_RNG_ReadyCallback ( <i>RNG_HandleTypeDef</i> * hrng)</b>
Function Description	Data Ready callback in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b> : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 38.2.6.5 HAL\_RNG\_ErrorCallback

Function Name	<b>void HAL_RNG_ErrorCallback ( <i>RNG_HandleTypeDef</i> * hrng)</b>
Function Description	RNG error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hrng</b> : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 38.2.7 Peripheral State functions

### 38.2.7.1 HAL\_RNG\_GetState

Function Name	<b>HAL_RNG_StateTypeDef HAL_RNG_GetState ( <i>RNG_HandleTypeDef</i> * hrng)</b>
Function Description	Returns the RNG state.
Parameters	<ul style="list-style-type: none"><li>• <b>hrng</b> : pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 38.3 RNG Firmware driver defines

### 38.3.1 RNG

RNG

*RNG\_Flag\_definition*

- #define: ***RNG\_FLAG\_DRDY* ((uint32\_t)0x0001)**

*Data ready*

- #define: ***RNG\_FLAG\_CECS* ((uint32\_t)0x0002)**

*Clock error current status*

- #define: **RNG\_FLAG\_SECS** ((*uint32\_t*)0x0004)

*Seed error current status*

***RNG\_Interrupt\_definition***

- #define: **RNG\_IT\_CEI** ((*uint32\_t*)0x20)

*Clock error interrupt*

- #define: **RNG\_IT\_SEI** ((*uint32\_t*)0x40)

*Seed error interrupt*

## 39 HAL RTC Generic Driver

### 39.1 RTC Firmware driver registers structures

#### 39.1.1 RTC\_HandleTypeDef

*RTC\_HandleTypeDef* is defined in the `stm32f4xx_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

#### 39.1.2 RTC\_InitTypeDef

*RTC\_InitTypeDef* is defined in the `stm32f4xx_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 [\*RTC\\_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\*](#)

### 39.1.3 RTC\_DateTypeDef

*RTC\_DateTypeDef* is defined in the `stm32f4xx_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

### 39.1.4 RTC\_TimeTypeDef

*RTC\_TimeTypeDef* is defined in the `stm32f4xx_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 DayLight Save Operation. 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](#)

### 39.1.5 RTC\_AlarmTypeDef

*RTC\_AlarmTypeDef* is defined in the `stm32f4xx_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\*](#)

### 39.1.6 RTC\_TypeDef

*RTC\_TypeDef* is defined in the stm32f439xx.h

#### Data Fields

- *\_\_IO uint32\_t TR*
- *\_\_IO uint32\_t DR*
- *\_\_IO uint32\_t CR*
- *\_\_IO uint32\_t ISR*
- *\_\_IO uint32\_t PRER*
- *\_\_IO uint32\_t WUTR*
- *\_\_IO uint32\_t CALIBR*
- *\_\_IO uint32\_t ALRMAR*
- *\_\_IO uint32\_t ALRMBR*
- *\_\_IO uint32\_t WPR*
- *\_\_IO uint32\_t SSR*
- *\_\_IO uint32\_t SHIFTR*
- *\_\_IO uint32\_t TSTR*
- *\_\_IO uint32\_t TSDR*
- *\_\_IO uint32\_t TSSSR*
- *\_\_IO uint32\_t CALR*
- *\_\_IO uint32\_t TAFCR*
- *\_\_IO uint32\_t ALRMASSR*
- *\_\_IO uint32\_t ALRMBSSR*
- *uint32\_t RESERVED7*
- *\_\_IO uint32\_t BKP0R*
- *\_\_IO uint32\_t BKP1R*
- *\_\_IO uint32\_t BKP2R*
- *\_\_IO uint32\_t BKP3R*
- *\_\_IO uint32\_t BKP4R*
- *\_\_IO uint32\_t BKP5R*
- *\_\_IO uint32\_t BKP6R*



- `__IO uint32_t BKP7R`
- `__IO uint32_t BKP8R`
- `__IO uint32_t BKP9R`
- `__IO uint32_t BKP10R`
- `__IO uint32_t BKP11R`
- `__IO uint32_t BKP12R`
- `__IO uint32_t BKP13R`
- `__IO uint32_t BKP14R`
- `__IO uint32_t BKP15R`
- `__IO uint32_t BKP16R`
- `__IO uint32_t BKP17R`
- `__IO uint32_t BKP18R`
- `__IO uint32_t BKP19R`

## Field Documentation

- `__IO uint32_t RTC_TypeDef::TR`
  - RTC time register, Address offset: 0x00
- `__IO uint32_t RTC_TypeDef::DR`
  - RTC date register, Address offset: 0x04
- `__IO uint32_t RTC_TypeDef::CR`
  - RTC control register, Address offset: 0x08
- `__IO uint32_t RTC_TypeDef::ISR`
  - RTC initialization and status register, Address offset: 0x0C
- `__IO uint32_t RTC_TypeDef::PRER`
  - RTC prescaler register, Address offset: 0x10
- `__IO uint32_t RTC_TypeDef::WUTR`
  - RTC wakeup timer register, Address offset: 0x14
- `__IO uint32_t RTC_TypeDef::CALIBR`
  - RTC calibration register, Address offset: 0x18
- `__IO uint32_t RTC_TypeDef::ALRMAR`
  - RTC alarm A register, Address offset: 0x1C
- `__IO uint32_t RTC_TypeDef::ALRMBR`
  - RTC alarm B register, Address offset: 0x20
- `__IO uint32_t RTC_TypeDef::WPR`
  - RTC write protection register, Address offset: 0x24
- `__IO uint32_t RTC_TypeDef::SSR`
  - RTC sub second register, Address offset: 0x28
- `__IO uint32_t RTC_TypeDef::SHIFTR`
  - RTC shift control register, Address offset: 0x2C
- `__IO uint32_t RTC_TypeDef::TSTR`
  - RTC time stamp time register, Address offset: 0x30
- `__IO uint32_t RTC_TypeDef::TSDR`
  - RTC time stamp date register, Address offset: 0x34
- `__IO uint32_t RTC_TypeDef::TSSSR`
  - RTC time-stamp sub second register, Address offset: 0x38
- `__IO uint32_t RTC_TypeDef::CALR`
  - RTC calibration register, Address offset: 0x3C
- `__IO uint32_t RTC_TypeDef::TAFCR`
  - RTC tamper and alternate function configuration register, Address offset: 0x40
- `__IO uint32_t RTC_TypeDef::ALRMASSR`

- RTC alarm A sub second register, Address offset: 0x44
- ***\_\_IO uint32\_t RTC\_TypeDef::ALRMBSSR***
  - RTC alarm B sub second register, Address offset: 0x48
- ***uint32\_t RTC\_TypeDef::RESERVED7***
  - Reserved, 0x4C
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP0R***
  - RTC backup register 1, Address offset: 0x50
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP1R***
  - RTC backup register 1, Address offset: 0x54
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP2R***
  - RTC backup register 2, Address offset: 0x58
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP3R***
  - RTC backup register 3, Address offset: 0x5C
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP4R***
  - RTC backup register 4, Address offset: 0x60
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP5R***
  - RTC backup register 5, Address offset: 0x64
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP6R***
  - RTC backup register 6, Address offset: 0x68
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP7R***
  - RTC backup register 7, Address offset: 0x6C
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP8R***
  - RTC backup register 8, Address offset: 0x70
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP9R***
  - RTC backup register 9, Address offset: 0x74
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP10R***
  - RTC backup register 10, Address offset: 0x78
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP11R***
  - RTC backup register 11, Address offset: 0x7C
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP12R***
  - RTC backup register 12, Address offset: 0x80
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP13R***
  - RTC backup register 13, Address offset: 0x84
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP14R***
  - RTC backup register 14, Address offset: 0x88
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP15R***
  - RTC backup register 15, Address offset: 0x8C
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP16R***
  - RTC backup register 16, Address offset: 0x90
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP17R***
  - RTC backup register 17, Address offset: 0x94
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP18R***
  - RTC backup register 18, Address offset: 0x98
- ***\_\_IO uint32\_t RTC\_TypeDef::BKP19R***
  - RTC backup register 19, Address offset: 0x9C

## 39.2 RTC Firmware driver API description

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

### 39.2.1 Backup Domain Operating Condition

The real-time clock (RTC), the RTC backup registers, and the backup SRAM (BKP SRAM) can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers, backup SRAM, and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

To allow the RTC operating even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

1. The RTC
2. The LSE oscillator
3. The backup SRAM when the low power backup regulator is enabled
4. PC13 to PC15 I/Os, plus PI8 I/O (when available)

When the backup domain is supplied by VDD (analog switch connected to VDD), the following pins 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\_AF1 pin
3. PI8 can be used as a GPIO or as the RTC\_AF2 pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following pins are available:

1. PC14 and PC15 can be used as LSE pins only
2. PC13 can be used as the RTC\_AF1 pin
3. PI8 can be used as the RTC\_AF2 pin

### 39.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC\_BDCR register to their reset values. The BKPSRAM is not affected by this reset. The only way to reset the BKPSRAM is through the Flash interface by requesting a protection level change from 1 to 0.

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.

### 39.2.3 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:

- Enable the Power Controller (PWR) APB1 interface clock using the \_\_PWR\_CLK\_ENABLE() function.
- Enable access to RTC domain using the HAL\_PWR\_EnableBkUpAccess() function.
- Select the RTC clock source using the \_\_HAL\_RCC\_RTC\_CONFIG() function.
- Enable RTC Clock using the \_\_HAL\_RCC\_RTC\_ENABLE() function.

### 39.2.4 How to use this 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.

### 39.2.5 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.

### 39.2.6 Initialization and de-initialization functions

This section provides 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 13-bit synchronous prescaler.
  - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC\_WPR.

3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC\_TR and RTC\_DR shadow registers. The `HAL_RTC_WaitForSynchro()` function implements the above software sequence (RSF clear and RSF check).
  - `HAL_RTC_Init()`
  - `HAL_RTC_DelInit()`
  - `HAL_RTC_MspInit()`
  - `HAL_RTC_MspDelInit()`

### 39.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

- `HAL_RTC_SetTime()`
- `HAL_RTC_GetTime()`
- `HAL_RTC_SetDate()`
- `HAL_RTC_GetDate()`

### 39.2.8 RTC Alarm functions

This section provides 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()`

### 39.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization
- `HAL_RTC_WaitForSynchro()`
- `RTC_EnterInitMode()`
- `RTC_ByteToBcd2()`
- `RTC_Bcd2ToByte()`

### 39.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state
- `HAL_RTC_GetState()`

### 39.2.11 Initialization and de-initialization functions

#### 39.2.11.1 HAL\_RTC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_RTC_Init ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	Initializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a <i>RTC_HandleTypeDef</i> structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 39.2.11.2 HAL\_RTC\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_RTC_DelInit ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	Deinitializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a <i>RTC_HandleTypeDef</i> structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• This function doesn't reset the RTC Backup Data registers.</li></ul>

#### 39.2.11.3 HAL\_RTC\_MspInit

Function Name	<b>void HAL_RTC_MspInit ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	Initializes the RTC MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a <i>RTC_HandleTypeDef</i> structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.11.4 HAL\_RTC\_MspDelInit

Function Name	<b>void HAL_RTC_MspDelInit ( <i>RTC_HandleTypeDef</i> * hrtc)</b>
Function Description	DeInitializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 39.2.12 RTC Time and Date functions

### 39.2.12.1 HAL\_RTC\_SetTime

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetTime ( <i>RTC_HandleTypeDef</i> * hrtc, <i>RTC_TimeTypeDef</i> * sTime, uint32_t Format)</b>
Function Description	Sets RTC current time.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sTime</b> : Pointer to Time structure</li> <li>• <b>Format</b> : Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>FORMAT_BIN</b> : Binary data format</li> <li>– <b>FORMAT_BCD</b> : BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.12.2 HAL\_RTC\_GetTime

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetTime (</b>
---------------	--

***RTC\_HandleTypeDef \* hrtc, RTC\_TimeTypeDef \* sTime,  
uint32\_t Format)***

Function Description	Gets RTC current time.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sTime</b> : Pointer to Time structure</li> <li>• <b>Format</b> : Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>FORMAT_BIN</b> : Binary data format</li> <li>– <b>FORMAT_BCD</b> : BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers.</li> </ul>

### 39.2.12.3 HAL\_RTC\_SetDate

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetDate ( <i>RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format</i>)</b>
Function Description	Sets RTC current date.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sDate</b> : Pointer to date structure</li> <li>• <b>Format</b> : specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>FORMAT_BIN</b> : Binary data format</li> <li>– <b>FORMAT_BCD</b> : BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.12.4 HAL\_RTC\_GetDate

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetDate ( <i>RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format</i>)</b>
---------------	---

Function Description	Gets RTC current date.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sDate</b> : Pointer to Date structure</li> <li>• <b>Format</b> : Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>FORMAT_BIN</b> : Binary data format</li> <li>– <b>FORMAT_BCD</b> : BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.13 RTC Alarm functions

#### 39.2.13.1 HAL\_RTC\_SetAlarm

Function Name	<code>HAL_StatusTypeDef HAL_RTC_SetAlarm (</code> <code>RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,</code> <code>uint32_t Format)</code>
Function Description	Sets the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sAlarm</b> : Pointer to Alarm structure</li> <li>• <b>Format</b> : Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>FORMAT_BIN</b> : Binary data format</li> <li>– <b>FORMAT_BCD</b> : BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 39.2.13.2 HAL\_RTC\_SetAlarm\_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (</code> <code>RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,</code> <code>uint32_t Format)</code>
Function Description	Sets the specified RTC Alarm with Interrupt.

Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>sAlarm</b> : Pointer to Alarm structure</li> <li><b>Format</b> : Specifies the format of the entered parameters. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>FORMAT_BIN</b> : Binary data format</li> <li>– <b>FORMAT_BCD</b> : BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 39.2.13.3 HAL\_RTC\_DeactivateAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t Alarm)</b>
Function Description	Deactivate the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>Alarm</b> : Specifies the Alarm. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RTC_ALARM_A</b> : AlarmA</li> <li>– <b>RTC_ALARM_B</b> : AlarmB</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 39.2.13.4 HAL\_RTC\_GetAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetAlarm (</b> <b>RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm,</b> <b>uint32_t Alarm, uint32_t Format)</b>
Function Description	Gets the RTC Alarm value and masks.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>sAlarm</b> : Pointer to Date structure</li> <li><b>Alarm</b> : Specifies the Alarm. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RTC_ALARM_A</b> : AlarmA</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>RTC_ALARM_B</i></b> : AlarmB</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>Format</b> : Specifies the format of the entered parameters. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <b><i>FORMAT_BIN</i></b> : Binary data format</li> <li>- <b><i>FORMAT_BCD</i></b> : BCD data format</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.13.5 HAL\_RTC\_AlarmIRQHandler

Function Name	<b>void HAL_RTC_AlarmIRQHandler ( <i>RTC_HandleTypeDef</i> * <b>hrtc</b>)</b>
Function Description	This function handles Alarm interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a <i>RTC_HandleTypeDef</i> structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.13.6 HAL\_RTC\_AlarmAEventCallback

Function Name	<b>void HAL_RTC_AlarmAEventCallback ( <i>RTC_HandleTypeDef</i> * <b>hrtc</b>)</b>
Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a <i>RTC_HandleTypeDef</i> structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.13.7 HAL\_RTC\_PollForAlarmAEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (</b> <i>RTC_HandleTypeDef * hrtc, uint32_t Timeout)</i>
Function Description	This function handles AlarmA Polling request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 39.2.14 Peripheral Control functions

#### 39.2.14.1 HAL\_RTC\_WaitForSynchro

Function Name	<b>HAL_StatusTypeDef HAL_RTC_WaitForSynchro (</b> <i>RTC_HandleTypeDef * hrtc)</i>
Function Description	Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The RTC Resynchronization mode is write protected, use the <code>_HAL_RTC_WRITEPROTECTION_DISABLE()</code> before calling this function.</li> <li>• 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.</li> </ul>

#### 39.2.14.2 RTC\_EnterInitMode

Function Name	<b>HAL_StatusTypeDef RTC_EnterInitMode (</b> <i>RTC_HandleTypeDef * hrtc)</i>
---------------	--

Function Description	Enters the RTC Initialization mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• The RTC Initialization mode is write protected, use the <code>_HAL_RTC_WRITEPROTECTION_DISABLE()</code> before calling this function.</li></ul>

### 39.2.14.3 RTC[ByteToBcd2]

Function Name	<b>uint8_t RTC[ByteToBcd2 ( uint8_t Value )]</b>
Function Description	Converts a 2 digit decimal to BCD format.
Parameters	<ul style="list-style-type: none"><li>• <b>Value</b> : Byte to be converted</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>Converted byte</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 39.2.14.4 RTC[Bcd2ToByte]

Function Name	<b>uint8_t RTC[Bcd2ToByte ( uint8_t Value )]</b>
Function Description	Converts from 2 digit BCD to Binary.
Parameters	<ul style="list-style-type: none"><li>• <b>Value</b> : BCD value to be converted</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>Converted word</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 39.2.15 Peripheral State functions

### 39.2.15.1 HAL\_RTC[GetState]

Function Name	<b>HAL_RTCStateTypeDef HAL_RTC_GetState (</b> <i>RTC_HandleTypeDef * hrtc)</i>
Function Description	Returns the RTC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 39.3 RTC Firmware driver defines

### 39.3.1 RTC

RTC

*RTC\_AlarmDateWeekDay\_Definitions*

- #define: **RTC\_ALARMDATEWEEKDAYSEL\_DATE** ((uint32\_t)0x00000000)
- #define: **RTC\_ALARMDATEWEEKDAYSEL\_WEEKDAY** ((uint32\_t)0x40000000)

*RTC\_AlarmMask\_Definitions*

- #define: **RTC\_ALARMMASK\_NONE** ((uint32\_t)0x00000000)
- #define: **RTC\_ALARMMASK\_DATEWEEKDAY RTC\_ALRMAR\_MSK4**
- #define: **RTC\_ALARMMASK\_HOURS RTC\_ALRMAR\_MSK3**
- #define: **RTC\_ALARMMASK\_MINUTES RTC\_ALRMAR\_MSK2**
- #define: **RTC\_ALARMMASK\_SECONDS RTC\_ALRMAR\_MSK1**

- #define: **RTC\_ALARMMASK\_ALL** ((*uint32\_t*)0x80808080)

#### ***RTC\_Alarms\_Definitions***

- #define: **RTC\_ALARM\_A RTC\_CR\_ALRAE**
- #define: **RTC\_ALARM\_B RTC\_CR\_ALRBE**

#### ***RTC\_Alarm\_Sub\_Seconds\_Masks\_Definitions***

- #define: **RTC\_ALARMSUBSECONDMASK\_ALL** ((*uint32\_t*)0x00000000)  
*All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm*
- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_1** ((*uint32\_t*)0x01000000)  
*SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.*
- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_2** ((*uint32\_t*)0x02000000)  
*SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared*
- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_3** ((*uint32\_t*)0x03000000)  
*SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared*
- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_4** ((*uint32\_t*)0x04000000)  
*SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared*
- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_5** ((*uint32\_t*)0x05000000)  
*SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared*
- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_6** ((*uint32\_t*)0x06000000)  
*SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared*
- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_7** ((*uint32\_t*)0x07000000)  
*SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared*
- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_8** ((*uint32\_t*)0x08000000)

*SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared*

- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_9** ((*uint32\_t*)0x09000000)

*SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared*

- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_10** ((*uint32\_t*)0xA0000000)

*SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared*

- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_11** ((*uint32\_t*)0xB0000000)

*SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared*

- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_12** ((*uint32\_t*)0xC0000000)

*SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared*

- #define: **RTC\_ALARMSUBSECONDMASK\_SS14\_13** ((*uint32\_t*)0xD0000000)

*SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared*

- #define: **RTC\_ALARMSUBSECONDMASK\_SS14** ((*uint32\_t*)0xE0000000)

*SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared*

- #define: **RTC\_ALARMSUBSECONDMASK\_None** ((*uint32\_t*)0xF0000000)

*SS[14:0] are compared and must match to activate alarm.*

#### ***RTC\_AM\_PM\_Definitions***

- #define: **RTC\_HOURFORMAT12\_AM** ((*uint8\_t*)0x00)

- #define: **RTC\_HOURFORMAT12\_PM** ((*uint8\_t*)0x40)

#### ***RTC\_DayLightSaving\_Definitions***

- #define: **RTC\_DAYLIGHTSAVING\_SUB1H** ((*uint32\_t*)0x00020000)

- #define: **RTC\_DAYLIGHTSAVING\_ADD1H** ((*uint32\_t*)0x00010000)

- #define: **RTC\_DAYLIGHTSAVING\_NONE** ((*uint32\_t*)0x00000000)

#### ***RTC\_Exported\_Constants***

- #define: **RTC\_TR\_RESERVED\_MASK** ((*uint32\_t*)0x007F7F7F)
- #define: **RTC\_DR\_RESERVED\_MASK** ((*uint32\_t*)0x00FFFF3F)
- #define: **RTC\_INIT\_MASK** ((*uint32\_t*)0xFFFFFFFF)
- #define: **RTC\_RSF\_MASK** ((*uint32\_t*)0xFFFFFFF5F)
- #define: **RTC\_FLAGS\_MASK** ((*uint32\_t*)(RTC\_FLAG\_TSOVF | RTC\_FLAG\_TSFI | RTC\_FLAG\_WUTF | \ RTC\_FLAG\_ALRBF | RTC\_FLAG\_ALRAF | RTC\_FLAG\_INITF | \ RTC\_FLAG\_RSF | RTC\_FLAG\_INITS | RTC\_FLAG\_WUTWF | \ RTC\_FLAG\_ALRBWF | RTC\_FLAG\_ALRAWF | RTC\_FLAG\_TAMP1F | \ RTC\_FLAG\_RECALPF | RTC\_FLAG\_SHPF))
- #define: **RTC\_TIMEOUT\_VALUE 1000**

#### ***RTC\_Flags\_Definitions***

- #define: **RTC\_FLAG\_RECALPF** ((*uint32\_t*)0x00010000)
- #define: **RTC\_FLAG\_TAMP2F** ((*uint32\_t*)0x00004000)
- #define: **RTC\_FLAG\_TAMP1F** ((*uint32\_t*)0x00002000)
- #define: **RTC\_FLAG\_TSOVF** ((*uint32\_t*)0x00001000)

- #define: ***RTC\_FLAG\_TSF*** ((*uint32\_t*)0x00000800)
- #define: ***RTC\_FLAG\_WUTF*** ((*uint32\_t*)0x00000400)
- #define: ***RTC\_FLAG\_ALRBF*** ((*uint32\_t*)0x00000200)
- #define: ***RTC\_FLAG\_ALRAF*** ((*uint32\_t*)0x00000100)
- #define: ***RTC\_FLAG\_INITF*** ((*uint32\_t*)0x00000040)
- #define: ***RTC\_FLAG\_RSF*** ((*uint32\_t*)0x00000020)
- #define: ***RTC\_FLAG\_INITS*** ((*uint32\_t*)0x00000010)
- #define: ***RTC\_FLAG\_SHPF*** ((*uint32\_t*)0x00000008)
- #define: ***RTC\_FLAG\_WUTWF*** ((*uint32\_t*)0x00000004)
- #define: ***RTC\_FLAG\_ALRBWF*** ((*uint32\_t*)0x00000002)
- #define: ***RTC\_FLAG\_ALRAWF*** ((*uint32\_t*)0x00000001)

#### ***RTC\_Hour\_Formats***

- #define: **RTC\_HOURFORMAT\_24** ((*uint32\_t*)0x00000000)
- #define: **RTC\_HOURFORMAT\_12** ((*uint32\_t*)0x00000040)

#### *RTC\_Input\_parameter\_format\_definitions*

- #define: **FORMAT\_BIN** ((*uint32\_t*)0x00000000)
- #define: **FORMAT\_BCD** ((*uint32\_t*)0x00000001)

#### *RTC Interrupts Definitions*

- #define: **RTC\_IT\_TS** ((*uint32\_t*)0x00008000)
- #define: **RTC\_IT\_WUT** ((*uint32\_t*)0x00004000)
- #define: **RTC\_IT\_ALRB** ((*uint32\_t*)0x00002000)
- #define: **RTC\_IT\_ALRA** ((*uint32\_t*)0x00001000)
- #define: **RTC\_IT\_TAMP** ((*uint32\_t*)0x00000004)
- #define: **RTC\_IT\_TAMP1** ((*uint32\_t*)0x00020000)
- #define: **RTC\_IT\_TAMP2** ((*uint32\_t*)0x00040000)

#### *RTC Month Date Definitions*

- #define: ***RTC\_MONTH\_JANUARY*** ((*uint8\_t*)0x01)
- #define: ***RTC\_MONTH\_FEBRUARY*** ((*uint8\_t*)0x02)
- #define: ***RTC\_MONTH\_MARCH*** ((*uint8\_t*)0x03)
- #define: ***RTC\_MONTH\_APRIl*** ((*uint8\_t*)0x04)
- #define: ***RTC\_MONTH\_MAY*** ((*uint8\_t*)0x05)
- #define: ***RTC\_MONTH\_JUNE*** ((*uint8\_t*)0x06)
- #define: ***RTC\_MONTH\_JULY*** ((*uint8\_t*)0x07)
- #define: ***RTC\_MONTH\_AUGUST*** ((*uint8\_t*)0x08)
- #define: ***RTC\_MONTH\_SEPTEMBER*** ((*uint8\_t*)0x09)
- #define: ***RTC\_MONTH\_OCTOBER*** ((*uint8\_t*)0x10)
- #define: ***RTC\_MONTH\_NOVEMBER*** ((*uint8\_t*)0x11)
- #define: ***RTC\_MONTH\_DECEMBER*** ((*uint8\_t*)0x12)

*RTC\_Output\_Polarity\_Definitions*

- #define: **RTC\_OUTPUT\_POLARITY\_HIGH** ((*uint32\_t*)0x00000000)
- #define: **RTC\_OUTPUT\_POLARITY\_LOW** ((*uint32\_t*)0x00100000)

*RTC\_Output\_selection\_Definitions*

- #define: **RTC\_OUTPUT\_DISABLE** ((*uint32\_t*)0x00000000)
- #define: **RTC\_OUTPUT\_ALARMA** ((*uint32\_t*)0x00200000)
- #define: **RTC\_OUTPUT\_ALARMB** ((*uint32\_t*)0x00400000)
- #define: **RTC\_OUTPUT\_WAKEUP** ((*uint32\_t*)0x00600000)

*RTC\_Output\_Type\_ALARM\_OUT*

- #define: **RTC\_OUTPUT\_TYPE\_OPENDRAIN** ((*uint32\_t*)0x00000000)
- #define: **RTC\_OUTPUT\_TYPE\_PUSHPULL** ((*uint32\_t*)0x00040000)

*RTC\_StoreOperation\_Definitions*

- #define: **RTC\_STOREOPERATION\_RESET** ((*uint32\_t*)0x00000000)
- #define: **RTC\_STOREOPERATION\_SET** ((*uint32\_t*)0x00040000)

*RTC\_WeekDay\_Definitions*

- #define: **RTC\_WEEKDAY\_MONDAY** ((*uint8\_t*)0x01)

- #define: ***RTC\_WEEKDAY\_TUESDAY ((uint8\_t)0x02)***
- #define: ***RTC\_WEEKDAY\_WEDNESDAY ((uint8\_t)0x03)***
- #define: ***RTC\_WEEKDAY\_THURSDAY ((uint8\_t)0x04)***
- #define: ***RTC\_WEEKDAY\_FRIDAY ((uint8\_t)0x05)***
- #define: ***RTC\_WEEKDAY\_SATURDAY ((uint8\_t)0x06)***
- #define: ***RTC\_WEEKDAY\_SUNDAY ((uint8\_t)0x07)***

## 40 HAL RTC Extension Driver

### 40.1 RTCEx Firmware driver registers structures

#### 40.1.1 RTC\_TamperTypeDef

*RTC\_TamperTypeDef* is defined in the `stm32f4xx_hal_rtc_ex.h`

##### Data Fields

- *uint32\_t Tamper*
- *uint32\_t PinSelection*
- *uint32\_t Trigger*
- *uint32\_t Filter*
- *uint32\_t SamplingFrequency*
- *uint32\_t PrechargeDuration*
- *uint32\_t TamperPullUp*
- *uint32\_tTimeStampOnTamperDetection*

##### 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::PinSelection*
  - Specifies the Tamper Pin. This parameter can be a value of [\*RTCEx\\_Tamper\\_Pins\\_Selection\*](#)
- *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\*](#)

## 40.2 RTCEx Firmware driver API description

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

### 40.2.1 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\_RTC\_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer in 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\_AFx trigger and enable 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.
- The TIMESTAMP alternate function can be mapped either to RTC\_AF1 (PC13) or RTC\_AF2 (PI8) depending on the value of TSINSEL bit in RTC\_TAFCR register. The corresponding pin is also selected by HAL\_RTC\_SetTimeStamp() or HAL\_RTC\_SetTimeStamp\_IT() 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 in interrupt mode using HAL\_RTC\_SetTamper\_IT() function.
- The TAMPER1 alternate function can be mapped either to RTC\_AF1 (PC13) or RTC\_AF2 (PI8) depending on the value of TAMP1INSEL bit in RTC\_TAFCR register. The corresponding pin is also selected by HAL\_RTC\_SetTamper() or 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.

## 40.2.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

- `HAL_RTCEEx_SetTimeStamp()`
- `HAL_RTCEEx_SetTimeStamp_IT()`
- `HAL_RTCEEx_DeactivateTimeStamp()`
- `HAL_RTCEEx_GetTimeStamp()`
- `HAL_RTCEEx_SetTamper()`
- `HAL_RTCEEx_SetTamper_IT()`
- `HAL_RTCEEx_DeactivateTamper()`
- `HAL_RTCEEx_TamperTimeStampIRQHandler()`
- `HAL_RTCEEx_TimeStampEventCallback()`
- `HAL_RTCEEx_Tamper1EventCallback()`
- `HAL_RTCEEx_Tamper2EventCallback()`
- `HAL_RTCEEx_PollForTimeStampEvent()`
- `HAL_RTCEEx_PollForTamper1Event()`
- `HAL_RTCEEx_PollForTamper2Event()`

## 40.2.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

- `HAL_RTCEEx_SetWakeUpTimer()`
- `HAL_RTCEEx_SetWakeUpTimer_IT()`
- `HAL_RTCEEx_DeactivateWakeUpTimer()`
- `HAL_RTCEEx_GetWakeUpTimer()`
- `HAL_RTCEEx_WakeUpTimerIRQHandler()`
- `HAL_RTCEEx_WakeUpTimerEventCallback()`
- `HAL_RTCEEx_PollForWakeUpTimerEvent()`

## 40.2.4 Extension Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.
- `HAL_RTCEEx_BKUPWrite()`
- `HAL_RTCEEx_BKUPRead()`
- `HAL_RTCEEx_SetCoarseCalib()`
- `HAL_RTCEEx_DeactivateCoarseCalib()`

- `HAL_RTCEx_SetSmoothCalib()`
- `HAL_RTCEx_SetSynchroShift()`
- `HAL_RTCEx_SetCalibrationOutPut()`
- `HAL_RTCEx_DeactivateCalibrationOutPut()`
- `HAL_RTCEx_SetRefClock()`
- `HAL_RTCEx_DeactivateRefClock()`
- `HAL_RTCEx_EnableBypassShadow()`
- `HAL_RTCEx_DisableBypassShadow()`

#### 40.2.5 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()`

#### 40.2.6 RTC TimeStamp and Tamper functions

##### 40.2.6.1 HAL\_RTCEx\_SetTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)</code>
Function Description	Sets TimeStamp.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>TimeStampEdge</b> : Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>RTC_TIMESTAMPEDGE_RISING</code> : the Time stamp event occurs on the rising edge of the related pin.</li> <li>- <code>RTC_TIMESTAMPEDGE_FALLING</code> : the Time stamp event occurs on the falling edge of the related pin.</li> </ul> </li> <li>• <b>RTC_TimeStampPin</b> : specifies the RTC TimeStamp Pin. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <code>RTC_TIMESTAMPPIN_PC13</code> : PC13 is selected as RTC TimeStamp Pin.</li> <li>- <code>RTC_TIMESTAMPPIN_PI8</code> : PI8 is selected as RTC TimeStamp Pin.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the TimeStamp feature.</li> </ul>

#### 40.2.6.2 HAL\_RTCEx\_SetTimeStamp\_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge,</code> <code>uint32_t RTC_TimeStampPin)</code>
Function Description	Sets TimeStamp with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>TimeStampEdge</b> : Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RTC_TIMESTAMPEDGE_RISING</b> : the Time stamp event occurs on the rising edge of the related pin.</li> <li>– <b>RTC_TIMESTAMPEDGE_FALLING</b> : the Time stamp event occurs on the falling edge of the related pin.</li> </ul> </li> <li>• <b>RTC_TimeStampPin</b> : Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>RTC_TIMESTAMPPIN_PC13</b> : PC13 is selected as RTC TimeStamp Pin.</li> <li>– <b>RTC_TIMESTAMPPIN_PI8</b> : PI8 is selected as RTC TimeStamp Pin.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the TimeStamp feature.</li> </ul>

#### 40.2.6.3 HAL\_RTCEx\_DeactivateTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (</code> <code>RTC_HandleTypeDef * hrtc)</code>
Function Description	Deactivates TimeStamp.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.6.4 HAL\_RTCEx\_GetTimeStamp

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * </code> <code>sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t </code> <code>Format)</code>
Function Description	Gets the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sTimeStamp</b> : Pointer to Time structure</li> <li>• <b>sTimeStampDate</b> : Pointer to Date structure</li> <li>• <b>Format</b> : 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</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.6.5 HAL\_RTCEx\_SetTamper

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTamper (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</code>
Function Description	Sets Tamper.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sTamper</b> : Pointer to Tamper Structure.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we disable the tamper interrupt for all tampers.</li> </ul>

#### 40.2.6.6 HAL\_RTCEx\_SetTamper\_IT

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (</code> <code>RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</code>
---------------	--

Function Description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>sTamper</b> : Pointer to RTC Tamper.</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>By calling this API we force the tamper interrupt for all tampers.</li> </ul>

#### 40.2.6.7 HAL\_RTCEx\_DeactivateTamper

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)</b>
Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>Tamper</b> : Selected tamper pin. This parameter can be RTC_Tamper_1 and/or RTC_TAMPER_2.</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 40.2.6.8 HAL\_RTCEx\_TamperTimeStampIRQHandler

Function Name	<b>void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)</b>
Function Description	This function handles TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 40.2.6.9 HAL\_RTCEx\_TimeStampEventCallback

Function Name	<b>void HAL_RTCEx_TimeStampEventCallback (</b> <i>RTC_HandleTypeDef * hrtc)</i>
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.6.10 HAL\_RTCEx\_Tamper1EventCallback

Function Name	<b>void HAL_RTCEx_Tamper1EventCallback (</b> <i>RTC_HandleTypeDef * hrtc)</i>
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.6.11 HAL\_RTCEx\_Tamper2EventCallback

Function Name	<b>void HAL_RTCEx_Tamper2EventCallback (</b> <i>RTC_HandleTypeDef * hrtc)</i>
Function Description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.6.12 HAL\_RTCEx\_PollForTimeStampEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles TimeStamp polling request.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.6.13 HAL\_RTCEx\_PollForTamper1Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles Tamper1 Polling.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.6.14 HAL\_RTCEx\_PollForTamper2Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles Tamper2 Polling.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li><li>• <b>Timeout</b> : Timeout duration</li></ul>

---

Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 40.2.7 RTC Wake-up functions

### 40.2.7.1 HAL\_RTCEx\_SetWakeUpTimer

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter,</b> <b>uint32_t WakeUpClock)</b>
Function Description	Sets wake up timer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>WakeUpCounter</b> : Wake up counter</li> <li>• <b>WakeUpClock</b> : Wake up clock</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 40.2.7.2 HAL\_RTCEx\_SetWakeUpTimer\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (</b> <b>RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter,</b> <b>uint32_t WakeUpClock)</b>
Function Description	Sets wake up timer with interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>WakeUpCounter</b> : Wake up counter</li> <li>• <b>WakeUpClock</b> : Wake up clock</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.7.3 HAL\_RTCEx\_DeactivateWakeUpTimer

Function Name	<code>uint32_t HAL_RTCEx_DeactivateWakeUpTimer ( RTC_HandleTypeDef * hrtc)</code>
Function Description	Deactivates wake up timer counter.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.7.4 HAL\_RTCEx\_GetWakeUpTimer

Function Name	<code>uint32_t HAL_RTCEx_GetWakeUpTimer ( RTC_HandleTypeDef * hrtc)</code>
Function Description	Gets wake up timer counter.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>Counter value</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.7.5 HAL\_RTCEx\_WakeUpTimerIRQHandler

Function Name	<code>void HAL_RTCEx_WakeUpTimerIRQHandler ( RTC_HandleTypeDef * hrtc)</code>
Function Description	This function handles Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 40.2.7.6 HAL\_RTCEx\_WakeUpTimerEventCallback

Function Name	<code>void HAL_RTCEx_WakeUpTimerEventCallback (</code> <code>RTC_HandleTypeDef * hrtc)</code>
Function Description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.7.7 HAL\_RTCEx\_PollForWakeUpTimerEvent

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (</code> <code>RTC_HandleTypeDef * hrtc, uint32_t Timeout)</code>
Function Description	This function handles Wake Up Timer Polling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 40.2.8 Extension Peripheral Control functions

#### 40.2.8.1 HAL\_RTCEx\_BKUPWrite

Function Name	<code>void HAL_RTCEx_BKUPWrite (</code> <code>RTC_HandleTypeDef * hrtc,</code> <code>uint32_t BackupRegister, uint32_t Data)</code>
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>BackupRegister</b> : RTC Backup data Register number. This</li> </ul>

---

	parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.
• <b>Data</b> :	Data to be written in the specified RTC Backup data register.
Return values	• None.
Notes	• None.

#### 40.2.8.2 HAL\_RTCEx\_BKUPRead

Function Name	<code>uint32_t HAL_RTCEx_BKUPRead ( <i>RTC_HandleTypeDef</i> * hrtc, uint32_t BackupRegister)</code>
Function Description	Reads data from the specified RTC Backup data Register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a <i>RTC_HandleTypeDef</i> structure that contains the configuration information for RTC.</li> <li>• <b>BackupRegister</b> : RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.</li> </ul>
Return values	• <b>Read value</b>
Notes	• None.

#### 40.2.8.3 HAL\_RTCEx\_SetCoarseCalib

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetCoarseCalib ( <i>RTC_HandleTypeDef</i> * hrtc, uint32_t CalibSign, uint32_t Value)</code>
Function Description	Sets the Coarse calibration parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a <i>RTC_HandleTypeDef</i> structure that contains the configuration information for RTC.</li> <li>• <b>CalibSign</b> : Specifies the sign of the coarse calibration value. This parameter can be one of the following values : <ul style="list-style-type: none"> <li>– <b>RTC_CALIBSIGN_POSITIVE</b> : The value sign is positive</li> <li>– <b>RTC_CALIBSIGN_NEGATIVE</b> : The value sign is negative</li> </ul> </li> <li>• <b>Value</b> : value of coarse calibration expressed in ppm (coded on 5 bits).</li> </ul>

---

Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>This Calibration value should be between 0 and 63 when using negative sign with a 2-ppm step.</li> <li>This Calibration value should be between 0 and 126 when using positive sign with a 4-ppm step.</li> </ul>

#### 40.2.8.4 HAL\_RTCEx\_DeactivateCoarseCalib

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateCoarseCalib (</b> <b><i>RTC_HandleTypeDef * hrtc</i></b> )
Function Description	Deactivates the Coarse calibration parameters.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 40.2.8.5 HAL\_RTCEx\_SetSmoothCalib

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (</b> <b><i>RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod,</i></b> <b><i>uint32_t SmoothCalibPlusPulses, uint32_t</i></b> <b><i>SmoothCalibMinusPulsesValue</i></b> )
Function Description	Sets the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>SmoothCalibPeriod</b> : Select the Smooth Calibration Period. This parameter can be one of the following values : <ul style="list-style-type: none"> <li><b>RTC_SMOOTHCALIB_PERIOD_32SEC</b> : The smooth calibration period is 32s.</li> <li><b>RTC_SMOOTHCALIB_PERIOD_16SEC</b> : The smooth calibration period is 16s.</li> <li><b>RTC_SMOOTHCALIB_PERIOD_8SEC</b> : The smooth calibration period is 8s.</li> </ul> </li> <li><b>SmoothCalibPlusPulses</b> : Select to Set or reset the CALP bit. This parameter can be one of the following values: <ul style="list-style-type: none"> <li><b>RTC_SMOOTHCALIB_PLUSPULSES_SET</b> : Add one RTCCCLK pulse every 2*11 pulses.</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>– <b><i>RTC_SMOOTHCALIB_PLUSPULSES_RESET</i></b> : No RTCCLK pulses are added.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.</li> </ul>

#### 40.2.8.6 HAL\_RTCEx\_SetSynchroShift

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (</b> <b><i>RTC_HandleTypeDef</i> * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)</b>
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>ShiftAdd1S</b> : Select to add or not 1 second to the time calendar. This parameter can be one of the following values : <ul style="list-style-type: none"> <li>– <b><i>RTC_SHIFTADD1S_SET</i></b> : Add one second to the clock calendar.</li> <li>– <b><i>RTC_SHIFTADD1S_RESET</i></b> : No effect.</li> </ul> </li> <li>• <b>ShiftSubFS</b> : Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When REFCKON is set, firmware must not write to Shift control register.</li> </ul>

#### 40.2.8.7 HAL\_RTCEx\_SetCalibrationOutPut

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (</b> <b><i>RTC_HandleTypeDef</i> * hrtc, uint32_t CalibOutput)</b>
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>CalibOutput</b> : Select the Calibration output Selection . This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b>RTC_CALIBOUTPUT_512HZ</b> : A signal has a regular waveform at 512Hz.</li> <li>– <b>RTC_CALIBOUTPUT_1HZ</b> : A signal has a regular waveform at 1Hz.</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 40.2.8.8 HAL\_RTCEx\_DeactivateCalibrationOutPut

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut ( RTC_HandleTypeDef * hrtc )</b>
Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 40.2.8.9 HAL\_RTCEx\_SetRefClock

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetRefClock ( RTC_HandleTypeDef * hrtc )</b>
Function Description	Enables the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 40.2.8.10 HAL\_RTCEx\_DeactivateRefClock

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (</b> <b><i>RTC_HandleTypeDef * hrtc</i></b> )
Function Description	Disable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 40.2.8.11 HAL\_RTCEx\_EnableBypassShadow

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (</b> <b><i>RTC_HandleTypeDef * hrtc</i></b> )
Function Description	Enables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.</li> </ul>

#### 40.2.8.12 HAL\_RTCEx\_DisableBypassShadow

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (</b> <b><i>RTC_HandleTypeDef * hrtc</i></b> )
Function Description	Disables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the Bypass Shadow is enabled the calendar value are</li> </ul>

---

taken directly from the Calendar counter.

## 40.2.9 Extended features functions

### 40.2.9.1 HAL\_RTCEx\_AlarmBEventCallback

Function Name	<code>void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)</code>
Function Description	Alarm B callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 40.2.9.2 HAL\_RTCEx\_PollForAlarmBEvent

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</code>
Function Description	This function handles AlarmB Polling request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b> : pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 40.3 RTCEx Firmware driver defines

### 40.3.1 RTCEx

RTCEx

*RTCEx\_Add\_1\_Second\_Parameter\_Definitions*

- #define: **RTC\_SHIFTADD1S\_RESET ((uint32\_t)0x00000000)**
- #define: **RTC\_SHIFTADD1S\_SET ((uint32\_t)0x80000000)**

***RTCEEx\_Backup\_Registers\_Definitions***

- #define: **RTC\_BKP\_DR0 ((uint32\_t)0x00000000)**
- #define: **RTC\_BKP\_DR1 ((uint32\_t)0x00000001)**
- #define: **RTC\_BKP\_DR2 ((uint32\_t)0x00000002)**
- #define: **RTC\_BKP\_DR3 ((uint32\_t)0x00000003)**
- #define: **RTC\_BKP\_DR4 ((uint32\_t)0x00000004)**
- #define: **RTC\_BKP\_DR5 ((uint32\_t)0x00000005)**
- #define: **RTC\_BKP\_DR6 ((uint32\_t)0x00000006)**
- #define: **RTC\_BKP\_DR7 ((uint32\_t)0x00000007)**
- #define: **RTC\_BKP\_DR8 ((uint32\_t)0x00000008)**
- #define: **RTC\_BKP\_DR9 ((uint32\_t)0x00000009)**

- #define: ***RTC\_BKP\_DR10 ((uint32\_t)0x0000000A)***
- #define: ***RTC\_BKP\_DR11 ((uint32\_t)0x0000000B)***
- #define: ***RTC\_BKP\_DR12 ((uint32\_t)0x0000000C)***
- #define: ***RTC\_BKP\_DR13 ((uint32\_t)0x0000000D)***
- #define: ***RTC\_BKP\_DR14 ((uint32\_t)0x0000000E)***
- #define: ***RTC\_BKP\_DR15 ((uint32\_t)0x0000000F)***
- #define: ***RTC\_BKP\_DR16 ((uint32\_t)0x00000010)***
- #define: ***RTC\_BKP\_DR17 ((uint32\_t)0x00000011)***
- #define: ***RTC\_BKP\_DR18 ((uint32\_t)0x00000012)***
- #define: ***RTC\_BKP\_DR19 ((uint32\_t)0x00000013)***

#### ***RTCEEx\_Calib\_Output\_selection\_Definitions***

- #define: ***RTC\_CALIBOUTPUT\_512HZ ((uint32\_t)0x00000000)***
- #define: ***RTC\_CALIBOUTPUT\_1HZ ((uint32\_t)0x00080000)***

***RTCEEx\_Digital\_Calibration\_Definitions***

- #define: ***RTC\_CALIBSIGN\_POSITIVE ((uint32\_t)0x00000000)***
  
- #define: ***RTC\_CALIBSIGN\_NEGATIVE ((uint32\_t)0x00000080)***

***RTCEEx\_Smooth\_calib\_period\_Definitions***

- #define: ***RTC\_SMOOTHCALIB\_PERIOD\_32SEC ((uint32\_t)0x00000000)***  
*If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK seconds*

- #define: ***RTC\_SMOOTHCALIB\_PERIOD\_16SEC ((uint32\_t)0x00002000)***  
*If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK seconds*

- #define: ***RTC\_SMOOTHCALIB\_PERIOD\_8SEC ((uint32\_t)0x00004000)***  
*If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK seconds*

***RTCEEx\_Smooth\_calib\_Plus\_pulses\_Definitions***

- #define: ***RTC\_SMOOTHCALIB\_PLUSPULSES\_SET ((uint32\_t)0x00008000)***  
*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*

- #define: ***RTC\_SMOOTHCALIB\_PLUSPULSES\_RESET ((uint32\_t)0x00000000)***  
*The number of RTCCLK pulses substituted during a 32-second window = CALM[8:0]*

***RTCEEx\_Tamper\_Filter\_Definitions***

- #define: ***RTC\_TAMPERFILTER\_DISABLE ((uint32\_t)0x00000000)***

*Tamper filter is disabled*

- #define: ***RTC\_TAMPERFILTER\_2SAMPLE ((uint32\_t)0x00000800)***

*Tamper is activated after 2 consecutive samples at the active level*

- #define: ***RTC\_TAMPERFILTER\_4SAMPLE ((uint32\_t)0x00001000)***

*Tamper is activated after 4 consecutive samples at the active level*

- #define: **RTC\_TAMPERFILTER\_8SAMPLE** ((uint32\_t)0x00001800)

*Tamper is activated after 8 consecutive samples at the active leve.*

#### ***RTCEEx\_Tamper\_Pins\_Definitions***

- #define: **RTC\_TAMPER\_1 RTC\_TAFCR\_TAMP1E**

- #define: **RTC\_TAMPER\_2 RTC\_TAFCR\_TAMP2E**

#### ***RTCEEx\_Tamper\_Pins\_Selection***

- #define: **RTC\_TAMPERPIN\_PC13** ((uint32\_t)0x00000000)

- #define: **RTC\_TAMPERPIN\_PI8** ((uint32\_t)0x00010000)

#### ***RTCEEx\_Tamper\_Pin\_Precharge\_Duration\_Definitions***

- #define: **RTC\_TAMPERPRECHARGEDURATION\_1RTCCLK** ((uint32\_t)0x00000000)

*Tamper pins are pre-charged before sampling during 1 RTCCLK cycle*

- #define: **RTC\_TAMPERPRECHARGEDURATION\_2RTCCLK** ((uint32\_t)0x00002000)

*Tamper pins are pre-charged before sampling during 2 RTCCLK cycles*

- #define: **RTC\_TAMPERPRECHARGEDURATION\_4RTCCLK** ((uint32\_t)0x00004000)

*Tamper pins are pre-charged before sampling during 4 RTCCLK cycles*

- #define: **RTC\_TAMPERPRECHARGEDURATION\_8RTCCLK** ((uint32\_t)0x00006000)

*Tamper pins are pre-charged before sampling during 8 RTCCLK cycles*

#### ***RTCEEx\_Tamper\_Pull\_UP\_Definitions***

- #define: **RTC\_TAMPER\_PULLUP\_ENABLE** ((uint32\_t)0x00000000)

*TimeStamp on Tamper Detection event saved*

- `#define: RTC_TAMPER_PULLUP_DISABLE  
((uint32_t)RTC_TAFCR_TAMPPUDIS)`

*TimeStamp on Tamper Detection event is not saved*

#### ***RTCEEx\_Tamper\_Sampling\_Frequencies\_Definitions***

- `#define: RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768  
((uint32_t)0x00000000)`

*Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768*

- `#define: RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384  
((uint32_t)0x00000100)`

*Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384*

- `#define: RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192  
((uint32_t)0x00000200)`

*Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192*

- `#define: RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096  
((uint32_t)0x00000300)`

*Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096*

- `#define: RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048  
((uint32_t)0x00000400)`

*Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048*

- `#define: RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024  
((uint32_t)0x00000500)`

*Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024*

- `#define: RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512  
((uint32_t)0x00000600)`

*Each of the tamper inputs are sampled with a frequency = RTCCLK / 512*

- `#define: RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256  
((uint32_t)0x00000700)`

*Each of the tamper inputs are sampled with a frequency = RTCCLK / 256*

#### ***RTCEEx\_Tamper\_TimeStampOnTamperDetection\_Definitions***

- `#define: RTC_TIMESTAMPONTAMPERDETECTION_ENABLE  
((uint32_t)RTC_TAFCR_TAMPTS)`

*TimeStamp on Tamper Detection event saved*

- #define: ***RTC\_TIMESTAMPONTAMPERDETECTION\_DISABLE***  
**((uint32\_t)0x00000000)**

*TimeStamp on Tamper Detection event is not saved*

#### ***RTCEEx\_Tamper\_Trigger\_Definitions***

- #define: ***RTC\_TAMPERTRIGGER\_RISINGEDGE*** ((*uint32\_t*)0x00000000)
- #define: ***RTC\_TAMPERTRIGGER\_FALLINGEDGE*** ((*uint32\_t*)0x00000002)
- #define: ***RTC\_TAMPERTRIGGER\_LOWLEVEL***  
***RTC\_TAMPERTRIGGER\_RISINGEDGE***
- #define: ***RTC\_TAMPERTRIGGER\_HIGHLEVEL***  
***RTC\_TAMPERTRIGGER\_FALLINGEDGE***

#### ***RTCEEx\_TimeStamp\_Pin\_Selection***

- #define: ***RTC\_TIMESTAMPPIN\_PC13*** ((*uint32\_t*)0x00000000)
- #define: ***RTC\_TIMESTAMPPIN\_PI8*** ((*uint32\_t*)0x00020000)

#### ***RTCEEx\_Time\_Stamp\_Edges\_definitions***

- #define: ***RTC\_TIMESTAMPEDGE\_RISING*** ((*uint32\_t*)0x00000000)
- #define: ***RTC\_TIMESTAMPEDGE\_FALLING*** ((*uint32\_t*)0x00000008)

#### ***RTCEEx\_Wakeup\_Timer\_Definitions***

- #define: ***RTC\_WAKEUPCLOCK\_RTCCLK\_DIV16*** ((*uint32\_t*)0x00000000)

- #define: ***RTC\_WAKEUPCLOCK\_RTCCLK\_DIV8 ((uint32\_t)0x00000001)***
- #define: ***RTC\_WAKEUPCLOCK\_RTCCLK\_DIV4 ((uint32\_t)0x00000002)***
- #define: ***RTC\_WAKEUPCLOCK\_RTCCLK\_DIV2 ((uint32\_t)0x00000003)***
- #define: ***RTC\_WAKEUPCLOCK\_CK\_SPRE\_16BITS ((uint32\_t)0x00000004)***
- #define: ***RTC\_WAKEUPCLOCK\_CK\_SPRE\_17BITS ((uint32\_t)0x00000006)***

## 41 HAL SAI Generic Driver

### 41.1 SAI Firmware driver registers structures

#### 41.1.1 SAI\_HandleTypeDef

**SAI\_HandleTypeDef** is defined in the `stm32f4xx_hal_sai.h`

##### Data Fields

- **SAI\_Block\_TypeDef \* Instance**
- **SAI\_InitTypeDef Init**
- **SAI\_FrameInitTypeDef FrameInit**
- **SAI\_SlotInitTypeDef SlotInit**
- **uint16\_t \* pTxBuffPtr**
- **uint16\_t TxXferSize**
- **uint16\_t TxXferCount**
- **uint16\_t \* pRxBuffPtr**
- **uint16\_t RxXferSize**
- **uint16\_t RxXferCount**
- **DMA\_HandleTypeDef \* hdmatx**
- **DMA\_HandleTypeDef \* hdmarx**
- **HAL\_LockTypeDef Lock**
- **\_\_IO HAL\_SAI\_StateTypeDef State**
- **\_\_IO uint32\_t ErrorCode**

##### Field Documentation

- **SAI\_Block\_TypeDef\* SAI\_HandleTypeDef::Instance**
  - SAI Blockx registers base address
- **SAI\_InitTypeDef SAI\_HandleTypeDef::Init**
  - SAI communication parameters
- **SAI\_FrameInitTypeDef SAI\_HandleTypeDef::FrameInit**
  - SAI Frame configuration parameters
- **SAI\_SlotInitTypeDef SAI\_HandleTypeDef::SlotInit**
  - SAI Slot configuration parameters
- **uint16\_t\* SAI\_HandleTypeDef::pTxBuffPtr**
  - Pointer to SAI Tx transfer Buffer
- **uint16\_t SAI\_HandleTypeDef::TxXferSize**
  - SAI Tx transfer size
- **uint16\_t SAI\_HandleTypeDef::TxXferCount**
  - SAI Tx transfer counter
- **uint16\_t\* SAI\_HandleTypeDef::pRxBuffPtr**
  - Pointer to SAI Rx transfer buffer
- **uint16\_t SAI\_HandleTypeDef::RxXferSize**
  - SAI Rx transfer size
- **uint16\_t SAI\_HandleTypeDef::RxXferCount**
  - SAI Rx transfer counter
- **DMA\_HandleTypeDef\* SAI\_HandleTypeDef::hdmatx**

- SAI Tx DMA handle parameters
- ***DMA\_HandleTypeDef\* SAI\_HandleTypeDef::hdmarx***
  - SAI Rx DMA handle parameters
- ***HAL\_LockTypeDef SAI\_HandleTypeDef::Lock***
  - SAI locking object
- ***\_IO HAL\_SAI\_StateTypeDef SAI\_HandleTypeDef::State***
  - SAI communication state
- ***\_IO uint32\_t SAI\_HandleTypeDef::ErrorCode***
  - SAI Error code

#### 41.1.2 SAI\_InitTypeDef

***SAI\_InitTypeDef*** is defined in the `stm32f4xx_hal_sai.h`

##### Data Fields

- ***uint32\_t Protocol***
- ***uint32\_t AudioMode***
- ***uint32\_t DataSize***
- ***uint32\_t FirstBit***
- ***uint32\_t ClockStrobing***
- ***uint32\_t Synchro***
- ***uint32\_t OutputDrive***
- ***uint32\_t NoDivider***
- ***uint32\_t FIFOThreshold***
- ***uint32\_t ClockSource***
- ***uint32\_t AudioFrequency***

##### Field Documentation

- ***uint32\_t SAI\_InitTypeDef::Protocol***
  - Specifies the SAI Block protocol. This parameter can be a value of [\*\*\*SAI\\_Block\\_Protocol\*\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::AudioMode***
  - Specifies the SAI Block audio Mode. This parameter can be a value of [\*\*\*SAI\\_Block\\_Mode\*\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::DataSize***
  - Specifies the SAI Block data size. This parameter can be a value of [\*\*\*SAI\\_Block\\_Data\\_Size\*\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::FirstBit***
  - Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [\*\*\*SAI\\_Block\\_MSB\\_LSB\\_transmission\*\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::ClockStrobing***
  - Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of [\*\*\*SAI\\_Block\\_Clock\\_Strobing\*\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::Synchro***
  - Specifies SAI Block synchronization. This parameter can be a value of [\*\*\*SAI\\_Block\\_Synchronization\*\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::OutputDrive***

- Specifies when SAI Block outputs are driven. This parameter can be a value of [\*\*SAI\\_Block\\_Output\\_Drive\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::NoDivider***
  - Specifies whether master clock will be divided or not. This parameter can be a value of [\*\*SAI\\_Block\\_NoDivider\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::FIFOThreshold***
  - Specifies SAI Block FIFO threshold. This parameter can be a value of [\*\*SAI\\_Block\\_Fifo\\_Threshold\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::ClockSource***
  - Specifies the SAI Block x Clock source. This parameter can be a value of [\*\*SAI\\_Clock\\_Source\*\*](#)
- ***uint32\_t SAI\_InitTypeDef::AudioFrequency***
  - Specifies the audio frequency sampling. This parameter can be a value of [\*\*SAI\\_Audio\\_Frequency\*\*](#)

#### 41.1.3 SAI\_FrameInitTypeDef

***SAI\_FrameInitTypeDef*** is defined in the `stm32f4xx_hal_sai.h`

##### Data Fields

- ***uint32\_t FrameLength***
- ***uint32\_t ActiveFrameLength***
- ***uint32\_t FSDefinition***
- ***uint32\_t FSPolarity***
- ***uint32\_t FSOFFset***

##### Field Documentation

- ***uint32\_t SAI\_FrameInitTypeDef::FrameLength***
  - Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min\_Data = 8 and Max\_Data = 256. : If master clock MCLK\_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.
- ***uint32\_t SAI\_FrameInitTypeDef::ActiveFrameLength***
  - Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min\_Data = 1 and Max\_Data = 128
- ***uint32\_t SAI\_FrameInitTypeDef::FSDefinition***
  - Specifies the Frame synchronization definition. This parameter can be a value of [\*\*SAI\\_Block\\_FS\\_Definition\*\*](#)
- ***uint32\_t SAI\_FrameInitTypeDef::FSPolarity***
  - Specifies the Frame synchronization Polarity. This parameter can be a value of [\*\*SAI\\_Block\\_FS\\_Polarity\*\*](#)
- ***uint32\_t SAI\_FrameInitTypeDef::FSOffset***
  - Specifies the Frame synchronization Offset. This parameter can be a value of [\*\*SAI\\_Block\\_FS\\_Offset\*\*](#)

#### 41.1.4 SAI\_SlotInitTypeDef

**SAI\_SlotInitTypeDef** is defined in the stm32f4xx\_hal\_sai.h

##### Data Fields

- *uint32\_t FirstBitOffset*
- *uint32\_t SlotSize*
- *uint32\_t SlotNumber*
- *uint32\_t SlotActive*

##### Field Documentation

- ***uint32\_t SAI\_SlotInitTypeDef::FirstBitOffset***
  - Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min\_Data = 0 and Max\_Data = 24
- ***uint32\_t SAI\_SlotInitTypeDef::SlotSize***
  - Specifies the Slot Size. This parameter can be a value of [SAI\\_Block\\_Slot\\_Size](#)
- ***uint32\_t SAI\_SlotInitTypeDef::SlotNumber***
  - Specifies the number of slot in the audio frame. This parameter must be a number between Min\_Data = 1 and Max\_Data = 16
- ***uint32\_t SAI\_SlotInitTypeDef::SlotActive***
  - Specifies the slots in audio frame that will be activated. This parameter can be a value of [SAI\\_Block\\_Slot\\_Active](#)

#### 41.1.5 SAI\_Block\_TypeDef

**SAI\_Block\_TypeDef** is defined in the stm32f439xx.h

##### Data Fields

- *\_\_IO uint32\_t CR1*
- *\_\_IO uint32\_t CR2*
- *\_\_IO uint32\_t FRCR*
- *\_\_IO uint32\_t SLOTR*
- *\_\_IO uint32\_t IMR*
- *\_\_IO uint32\_t SR*
- *\_\_IO uint32\_t CLRFR*
- *\_\_IO uint32\_t DR*

##### Field Documentation

- ***\_\_IO uint32\_t SAI\_Block\_TypeDef::CR1***
  - SAI block x configuration register 1, Address offset: 0x04
- ***\_\_IO uint32\_t SAI\_Block\_TypeDef::CR2***
  - SAI block x configuration register 2, Address offset: 0x08
- ***\_\_IO uint32\_t SAI\_Block\_TypeDef::FRCR***
  - SAI block x frame configuration register, Address offset: 0x0C

- `__IO uint32_t SAI_Block_TypeDef::SLOTR`
  - SAI block x slot register, Address offset: 0x10
- `__IO uint32_t SAI_Block_TypeDef::IMR`
  - SAI block x interrupt mask register, Address offset: 0x14
- `__IO uint32_t SAI_Block_TypeDef::SR`
  - SAI block x status register, Address offset: 0x18
- `__IO uint32_t SAI_Block_TypeDef::CLRFR`
  - SAI block x clear flag register, Address offset: 0x1C
- `__IO uint32_t SAI_Block_TypeDef::DR`
  - SAI block x data register, Address offset: 0x20

#### 41.1.6 SAI\_TypeDef

`SAI_TypeDef` is defined in the `stm32f439xx.h`

##### Data Fields

- `__IO uint32_t GCR`

##### Field Documentation

- `__IO uint32_t SAI_TypeDef::GCR`
  - SAI global configuration register, Address offset: 0x00

### 41.2 SAI Firmware driver API description

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

#### 41.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a `SAI_HandleTypeDef` handle structure.
2. Initialize the SAI low level resources by implementing the `HAL_SAI_MspInit()` API:
  - a. Enable the SAI interface clock.
  - b. SAI pins configuration:
    - Enable the clock for the SAI GPIOs.
    - Configure these SAI pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (`HAL_SAI_Transmit_IT()` and `HAL_SAI_Receive_IT()` APIs):
    - Configure the SAI interrupt priority.
    - Enable the NVIC SAI IRQ handle.
  - d. DMA Configuration if you need to use DMA process (`HAL_SAI_Transmit_DMA()` and `HAL_SAI_Receive_DMA()` APIs):
    - Declare a DMA handle structure for the Tx/Rx stream.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.

- Configure the DMA Tx/Rx Stream.
  - Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. Program the SAI Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL\_SAI\_Init() function. The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros \_\_SAI\_ENABLE\_IT() and \_\_SAI\_DISABLE\_IT() inside the transmit and receive process.

 Make sure that either:

- I2S PLL is configured or
- SAI PLL is configured or
- External clock source is configured after setting correctly the define constant EXTERNAL\_CLOCK\_VALUE in the stm32f4xx\_hal\_conf.h file.

 In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.

 In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.

 It is mandatory to respect the following conditions in order to avoid bad SAI behavior:

- First bit Offset <= (SLOT size - Data size)
- Data size <= SLOT size
- Number of SLOT x SLOT size = Frame length
- The number of slots should be even when SAI\_FS\_CHANNEL\_IDENTIFICATION is selected.

Three operation modes are available within this driver :

### **Polling mode IO operation**

- Send an amount of data in blocking mode using HAL\_SAI\_Transmit()
- Receive an amount of data in blocking mode using HAL\_SAI\_Receive()

### **Interrupt mode IO operation**

- Send an amount of data in non blocking mode using HAL\_SAI\_Transmit\_IT()
- At transmission end of transfer HAL\_SAI\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SAI\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_SAI\_Receive\_IT()
- At reception end of transfer HAL\_SAI\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SAI\_RxCpltCallback

- In case of transfer Error, HAL\_SAI\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SAI\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_SAI\_Transmit\_DMA()
- At transmission end of transfer HAL\_SAI\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SAI\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_SAI\_Receive\_DMA()
- At reception end of transfer HAL\_SAI\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SAI\_RxCpltCallback
- In case of transfer Error, HAL\_SAI\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SAI\_ErrorCallback
- Pause the DMA Transfer using HAL\_SAI\_DMAPause()
- Resume the DMA Transfer using HAL\_SAI\_DMAResume()
- Stop the DMA Transfer using HAL\_SAI\_DMAStop()

### SPI HAL driver macros list

Below the list of most used macros in USART HAL driver :

- \_\_HAL\_SAI\_ENABLE: Enable the SAI peripheral
- \_\_HAL\_SAI\_DISABLE: Disable the SAI peripheral
- \_\_HAL\_SAI\_ENABLE\_IT : Enable the specified SAI interrupts
- \_\_HAL\_SAI\_DISABLE\_IT : Disable the specified SAI interrupts
- \_\_HAL\_SAI\_GET\_IT\_SOURCE: Check if the specified SAI interrupt source is enabled or disabled
- \_\_HAL\_SAI\_GET\_FLAG: Check whether the specified SAI flag is set or not

## 41.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SAIx peripheral:

- User must implement HAL\_SAI\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function HAL\_SAI\_Init() to configure the selected device with the selected configuration:
  - Mode (Master/slave TX/RX)
  - Protocol
  - Data Size
  - MCLK Output
  - Audio frequency
  - FIFO Threshold
  - Frame Config
  - Slot Config
- Call the function HAL\_SAI\_DeInit() to restore the default configuration of the selected SAI peripheral.
- [\*\*HAL\\_SAI\\_Init\(\)\*\*](#)
- [\*\*HAL\\_SAI\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_SAI\\_MspInit\(\)\*\*](#)

- [\*HAL\\_SAI\\_MspDelInit\(\)\*](#)

### 41.2.3 IO operation functions

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

- There are two modes of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SAI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- Blocking mode functions are :
  - [\*HAL\\_SAI\\_Transmit\(\)\*](#)
  - [\*HAL\\_SAI\\_Receive\(\)\*](#)
  - [\*HAL\\_SAI\\_TransmitReceive\(\)\*](#)
- Non Blocking mode functions with Interrupt are :
  - [\*HAL\\_SAI\\_Transmit\\_IT\(\)\*](#)
  - [\*HAL\\_SAI\\_Receive\\_IT\(\)\*](#)
  - [\*HAL\\_SAI\\_TransmitReceive\\_IT\(\)\*](#)
- Non Blocking mode functions with DMA are :
  - [\*HAL\\_SAI\\_Transmit\\_DMA\(\)\*](#)
  - [\*HAL\\_SAI\\_Receive\\_DMA\(\)\*](#)
  - [\*HAL\\_SAI\\_TransmitReceive\\_DMA\(\)\*](#)
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - [\*HAL\\_SAI\\_TxCpltCallback\(\)\*](#)
  - [\*HAL\\_SAI\\_RxCpltCallback\(\)\*](#)
  - [\*HAL\\_SAI\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_SAI\\_Transmit\(\)\*](#)
- [\*HAL\\_SAI\\_Receive\(\)\*](#)
- [\*HAL\\_SAI\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_SAI\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_SAI\\_DMAPause\(\)\*](#)
- [\*HAL\\_SAI\\_DMAResume\(\)\*](#)
- [\*HAL\\_SAI\\_DMAStop\(\)\*](#)
- [\*HAL\\_SAI\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_SAI\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_SAI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SAI\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_SAI\\_TxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SAI\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_SAI\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SAI\\_ErrorCallback\(\)\*](#)

### 41.2.4 Peripheral State and Errors functions

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

- [\*HAL\\_SAI\\_GetState\(\)\*](#)
- [\*HAL\\_SAI\\_GetError\(\)\*](#)

## 41.2.5 Initialization and de-initialization functions

### 41.2.5.1 HAL\_SAI\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Init ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	Initializes the SAI according to the specified parameters in the SAI_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 41.2.5.2 HAL\_SAI\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DelInit ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	Deinitializes the SAI peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 41.2.5.3 HAL\_SAI\_MspInit

Function Name	<b>void HAL_SAI_MspInit ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	SAI MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.5.4 HAL\_SAI\_MspDelInit

Function Name	<b>void HAL_SAI_MspDelInit ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	SAI MSP DelInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 41.2.6 IO operation functions

#### 41.2.6.1 HAL\_SAI\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Transmit ( <i>SAI_HandleTypeDef</i> * hsai, uint16_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmits an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li><li>• <b>pData</b> : Pointer to data buffer</li><li>• <b>Size</b> : Amount of data to be sent</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.6.2 HAL\_SAI\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Receive ( <i>SAI_HandleTypeDef</i> * hsai, uint16_t * pData, uint16_t Size, uint32_t Timeout)</b>
---------------	--

---

Function Description	Receives an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.6.3 HAL\_SAI\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Transmit_IT (</b> <b>SAI_HandleTypeDef * hsai, uint16_t * pData, uint16_t Size)</b>
Function Description	Transmits an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.6.4 HAL\_SAI\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Receive_IT (</b> <b>SAI_HandleTypeDef * hsai, uint16_t * pData, uint16_t Size)</b>
Function Description	Receives an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.6.5 HAL\_SAI\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DMAPause (</b> <b><i>SAI_HandleTypeDef * hsai</i></b> )
Function Description	Pauses the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"><li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.6.6 HAL\_SAI\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DMAResume (</b> <b><i>SAI_HandleTypeDef * hsai</i></b> )
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"><li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.6.7 HAL\_SAI\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DMAStop (</b> <b><i>SAI_HandleTypeDef * hsai</i></b> )
Function Description	Stops the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"><li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.6.8 HAL\_SAI\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Transmit_DMA (</b> <b><i>SAI_HandleTypeDef</i> * hsai, uint16_t * pData, uint16_t Size)</b>
Function Description	Transmits an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.6.9 HAL\_SAI\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Receive_DMA (</b> <b><i>SAI_HandleTypeDef</i> * hsai, uint16_t * pData, uint16_t Size)</b>
Function Description	Receives an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 41.2.6.10 HAL\_SAI\_IRQHandler

Function Name	<b>void HAL_SAI_IRQHandler (</b> <b><i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	This function handles SAI interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that</li> </ul>

contains the configuration information for SAI module.

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul> |
| Notes         | <ul style="list-style-type: none"><li>• None.</li></ul>             |

#### 41.2.6.11 HAL\_SAI\_TxCpltCallback

Function Name	<b>void HAL_SAI_TxCpltCallback ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.6.12 HAL\_SAI\_TxHalfCpltCallback

Function Name	<b>void HAL_SAI_TxHalfCpltCallback ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	Tx Transfer Half completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hsai</b> : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.6.13 HAL\_SAI\_RxCpltCallback

Function Name	<b>void HAL_SAI_RxCpltCallback ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	Rx Transfer completed callbacks.

Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.6.14 HAL\_SAI\_RxHalfCpltCallback

Function Name	<b>void HAL_SAI_RxHalfCpltCallback ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	Rx Transfer half completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.6.15 HAL\_SAI\_ErrorCallback

Function Name	<b>void HAL_SAI_ErrorCallback ( <i>SAI_HandleTypeDef</i> * hsai)</b>
Function Description	SAI error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 41.2.7 Peripheral State functions

#### 41.2.7.1 HAL\_SAI\_GetState

Function Name	<code>HAL_SAI_StateTypeDef HAL_SAI_GetState (</code> <code>SAI_HandleTypeDef * hsai)</code>
Function Description	Returns the SAI state.
Parameters	<ul style="list-style-type: none"><li>• <code>hsai</code> : pointer to a <code>SAI_HandleTypeDef</code> structure that contains the configuration information for SAI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 41.2.7.2 HAL\_SAI\_GetError

Function Name	<code>uint32_t HAL_SAI_GetError (</code> <code>SAI_HandleTypeDef * hsai)</code>
Function Description	Return the SAI error code.
Parameters	<ul style="list-style-type: none"><li>• <code>hsai</code> : pointer to a <code>SAI_HandleTypeDef</code> structure that contains the configuration information for the specified SAI Block.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>SAI Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 41.3 SAI Firmware driver defines

### 41.3.1 SAI

SAI

*SAI\_Audio\_Frequency*

- #define: `SAI_AUDIO_FREQUENCY_192K ((uint32_t)192000)`
- #define: `SAI_AUDIO_FREQUENCY_96K ((uint32_t)96000)`

- #define: ***SAI\_AUDIO\_FREQUENCY\_48K*** ((*uint32\_t*)48000)
  - #define: ***SAI\_AUDIO\_FREQUENCY\_44K*** ((*uint32\_t*)44100)
  - #define: ***SAI\_AUDIO\_FREQUENCY\_32K*** ((*uint32\_t*)32000)
  - #define: ***SAI\_AUDIO\_FREQUENCY\_22K*** ((*uint32\_t*)22050)
  - #define: ***SAI\_AUDIO\_FREQUENCY\_16K*** ((*uint32\_t*)16000)
  - #define: ***SAI\_AUDIO\_FREQUENCY\_11K*** ((*uint32\_t*)11025)
  - #define: ***SAI\_AUDIO\_FREQUENCY\_8K*** ((*uint32\_t*)8000)
- 
- SAI\_Block\_Clock\_Strobing***
- #define: ***SAI\_CLOCKSTROBING\_FALLINGEDGE*** ((*uint32\_t*)0x00000000)
  - #define: ***SAI\_CLOCKSTROBING\_RISINGEDGE*** ((*uint32\_t*)*SAI\_xCR1\_CKSTR*)
- 
- SAI\_Block\_Compadding\_Mode***
- #define: ***SAI\_NOCOMPANDING*** ((*uint32\_t*)0x00000000)
  - #define: ***SAI\_ULAW\_1CPL\_COMPANDING*** ((*uint32\_t*)0x00008000)

- #define: **SAI\_ALAW\_1CPL\_COMPANDING** ((*uint32\_t*)0x0000C000)
- #define: **SAI\_ULAW\_2CPL\_COMPANDING** ((*uint32\_t*)0x0000A000)
- #define: **SAI\_ALAW\_2CPL\_COMPANDING** ((*uint32\_t*)0x0000E000)

#### ***SAI\_Block\_Data\_Size***

- #define: **SAI\_DATASIZE\_8** ((*uint32\_t*)0x00000040)
- #define: **SAI\_DATASIZE\_10** ((*uint32\_t*)0x00000060)
- #define: **SAI\_DATASIZE\_16** ((*uint32\_t*)0x00000080)
- #define: **SAI\_DATASIZE\_20** ((*uint32\_t*)0x000000A0)
- #define: **SAI\_DATASIZE\_24** ((*uint32\_t*)0x000000C0)
- #define: **SAI\_DATASIZE\_32** ((*uint32\_t*)0x000000E0)

#### ***SAI\_Block\_Fifo\_Status\_Level***

- #define: **SAI\_FIFOStatus\_Empty** ((*uint32\_t*)0x00000000)
- #define: **SAI\_FIFOStatus\_Less1QuarterFull** ((*uint32\_t*)0x00010000)
- #define: **SAI\_FIFOStatus\_1QuarterFull** ((*uint32\_t*)0x00020000)

- #define: **SAI\_FIFOStatus\_HalfFull** ((*uint32\_t*)0x00030000)
- #define: **SAI\_FIFOStatus\_3QuartersFull** ((*uint32\_t*)0x00040000)
- #define: **SAI\_FIFOStatus\_Full** ((*uint32\_t*)0x00050000)

#### ***SAI\_Block\_Fifo\_Threshold***

- #define: **SAI\_FIFOTHRESHOLD\_EMPTY** ((*uint32\_t*)0x00000000)
- #define: **SAI\_FIFOTHRESHOLD\_1QF** ((*uint32\_t*)0x00000001)
- #define: **SAI\_FIFOTHRESHOLD\_HF** ((*uint32\_t*)0x00000002)
- #define: **SAI\_FIFOTHRESHOLD\_3QF** ((*uint32\_t*)0x00000003)
- #define: **SAI\_FIFOTHRESHOLD\_FULL** ((*uint32\_t*)0x00000004)

#### ***SAI\_Block\_Flags\_Definition***

- #define: **SAI\_FLAG\_OVRUDR** ((*uint32\_t*)**SAI\_xSR\_OVRUDR**)
- #define: **SAI\_FLAG\_MUTEDET** ((*uint32\_t*)**SAI\_xSR\_MUTEDET**)
- #define: **SAI\_FLAG\_WCKCFG** ((*uint32\_t*)**SAI\_xSR\_WCKCFG**)

- #define: ***SAI\_FLAG\_FREQ ((uint32\_t)SAI\_xSR\_FREQ)***
- #define: ***SAI\_FLAG\_CNRDY ((uint32\_t)SAI\_xSR\_CNRDY)***
- #define: ***SAI\_FLAG\_AFSDET ((uint32\_t)SAI\_xSR\_AFSDET)***
- #define: ***SAI\_FLAG\_LFSDET ((uint32\_t)SAI\_xSR\_LFSDET)***

#### ***SAI\_Block\_FS\_Definition***

- #define: ***SAI\_FS\_STARTFRAME ((uint32\_t)0x00000000)***
- #define: ***SAI\_FS\_CHANNEL\_IDENTIFICATION ((uint32\_t)SAI\_xFRCR\_FSDEF)***

#### ***SAI\_Block\_FS\_Offset***

- #define: ***SAI\_FS\_FIRSTBIT ((uint32\_t)0x00000000)***
- #define: ***SAI\_FS\_BEFOREFIRSTBIT ((uint32\_t)SAI\_xFRCR\_FSOFF)***

#### ***SAI\_Block\_FS\_Polarity***

- #define: ***SAI\_FS\_ACTIVE\_LOW ((uint32\_t)0x00000000)***
- #define: ***SAI\_FS\_ACTIVE\_HIGH ((uint32\_t)SAI\_xFRCR\_FSPO)***

#### ***SAI\_Block\_Interrupts\_Definition***

- #define: ***SAI\_IT\_OVRUDR ((uint32\_t)SAI\_xIMR\_OVRUDRIE)***

- #define: ***SAI\_IT\_MUTEDET ((uint32\_t)SAI\_xIMR\_MUTEDETIE)***
- #define: ***SAI\_IT\_WCKCFG ((uint32\_t)SAI\_xIMR\_WCKCFGIE)***
- #define: ***SAI\_IT\_FREQ ((uint32\_t)SAI\_xIMR\_FREQIE)***
- #define: ***SAI\_IT\_CNRDY ((uint32\_t)SAI\_xIMR\_CNRDYIE)***
- #define: ***SAI\_IT\_AFSDET ((uint32\_t)SAI\_xIMR\_AFSDETIE)***
- #define: ***SAI\_IT\_LFSDET ((uint32\_t)SAI\_xIMR\_LFSDETIE)***

#### ***SAI\_Block\_Mode***

- #define: ***SAI\_MODEMASTER\_TX ((uint32\_t)0x00000000)***
- #define: ***SAI\_MODEMASTER\_RX ((uint32\_t)0x00000001)***
- #define: ***SAI\_MODESLAVE\_TX ((uint32\_t)0x00000002)***
- #define: ***SAI\_MODESLAVE\_RX ((uint32\_t)0x00000003)***

#### ***SAI\_Block\_MSB\_LSB\_transmission***

- #define: ***SAI\_FIRSTBIT\_MSB ((uint32\_t)0x00000000)***

- #define: **SAI\_FIRSTBIT\_LSB** ((uint32\_t)SAI\_xCR1\_LSBFIRST)

#### ***SAI\_Block\_Mute\_Value***

- #define: **SAI\_ZERO\_VALUE** ((uint32\_t)0x00000000)
- #define: **SAI\_LAST\_SENT\_VALUE** ((uint32\_t)SAI\_xCR2\_MUTEVAL)

#### ***SAI\_Block\_NoDivider***

- #define: **SAI\_MASTERDIVIDER\_ENABLED** ((uint32\_t)0x00000000)
- #define: **SAI\_MASTERDIVIDER\_DISABLED** ((uint32\_t)SAI\_xCR1\_NODIV)

#### ***SAI\_Block\_Output\_Drive***

- #define: **SAI\_OUTPUTDRIVE\_DISABLED** ((uint32\_t)0x00000000)
- #define: **SAI\_OUTPUTDRIVE\_ENABLED** ((uint32\_t)SAI\_xCR1\_OUTDRV)

#### ***SAI\_Block\_Protocol***

- #define: **SAI\_FREE\_PROTOCOL** ((uint32\_t)0x00000000)
- #define: **SAI\_AC97\_PROTOCOL** ((uint32\_t)SAI\_xCR1\_PRTCFG\_1)

#### ***SAI\_Block\_Slot\_Active***

- #define: **SAI\_SLOT\_NOTACTIVE** ((uint32\_t)0x00000000)

- #define: ***SAI\_SLOTACTIVE\_0*** ((*uint32\_t*)0x00010000)
- #define: ***SAI\_SLOTACTIVE\_1*** ((*uint32\_t*)0x00020000)
- #define: ***SAI\_SLOTACTIVE\_2*** ((*uint32\_t*)0x00040000)
- #define: ***SAI\_SLOTACTIVE\_3*** ((*uint32\_t*)0x00080000)
- #define: ***SAI\_SLOTACTIVE\_4*** ((*uint32\_t*)0x00100000)
- #define: ***SAI\_SLOTACTIVE\_5*** ((*uint32\_t*)0x00200000)
- #define: ***SAI\_SLOTACTIVE\_6*** ((*uint32\_t*)0x00400000)
- #define: ***SAI\_SLOTACTIVE\_7*** ((*uint32\_t*)0x00800000)
- #define: ***SAI\_SLOTACTIVE\_8*** ((*uint32\_t*)0x01000000)
- #define: ***SAI\_SLOTACTIVE\_9*** ((*uint32\_t*)0x02000000)
- #define: ***SAI\_SLOTACTIVE\_10*** ((*uint32\_t*)0x04000000)
- #define: ***SAI\_SLOTACTIVE\_11*** ((*uint32\_t*)0x08000000)

- #define: **SAI\_SLOTACTIVE\_12** ((*uint32\_t*)0x10000000)
- #define: **SAI\_SLOTACTIVE\_13** ((*uint32\_t*)0x20000000)
- #define: **SAI\_SLOTACTIVE\_14** ((*uint32\_t*)0x40000000)
- #define: **SAI\_SLOTACTIVE\_15** ((*uint32\_t*)0x80000000)
- #define: **SAI\_SLOTACTIVE\_ALL** ((*uint32\_t*)0xFFFF0000)

#### ***SAI\_Block\_Slot\_Size***

- #define: **SAI\_SLOTSIZE\_DATASIZE** ((*uint32\_t*)0x00000000)
- #define: **SAI\_SLOTSIZE\_16B** ((*uint32\_t*)**SAI\_xSLOTR\_SLOTZ\_0**)
- #define: **SAI\_SLOTSIZE\_32B** ((*uint32\_t*)**SAI\_xSLOTR\_SLOTZ\_1**)

#### ***SAI\_Block\_Synchronization***

- #define: **SAI\_ASYNCROUS** ((*uint32\_t*)0x00000000)
- #define: **SAI\_SYNCHRONOUS** ((*uint32\_t*)**SAI\_xCR1\_SYNCEN\_0**)

#### ***SAI\_Clock\_Source***

- #define: **SAI\_CLKSOURCE\_PLLSAI** ((*uint32\_t*)**RCC\_SAIACLKSOURCE\_PLLSAI**)

- #define: **SAI\_CLKSOURCE\_PLLI2S ((uint32\_t)RCC\_SAIACLKSOURCE\_PLLI2S)**
- #define: **SAI\_CLKSOURCE\_EXT ((uint32\_t)RCC\_SAIACLKSOURCE\_EXT)**

#### ***SAI\_Mono\_Stereo\_Mode***

- #define: **SAI\_MONOMODE ((uint32\_t)SAI\_xCR1\_MONO)**
- #define: **SAI\_STREOMODE ((uint32\_t)0x00000000)**

#### ***SAI\_TRIState\_Management***

- #define: **SAI\_OUTPUT\_NOTRELEASED ((uint32\_t)0x00000000)**
- #define: **SAI\_OUTPUT\_RELEASED ((uint32\_t)SAI\_xCR2\_TRIS)**

## 42 HAL SMARTCARD Generic Driver

### 42.1 SMARTCARD Firmware driver registers structures

#### 42.1.1 SMARTCARD\_HandleTypeDef

*SMARTCARD\_HandleTypeDef* is defined in the `stm32f4xx_hal_smartcard.h`

##### Data Fields

- `USART_TypeDef * Instance`
- `SMARTCARD_InitTypeDef Init`
- `uint8_t * pTxBuffPtr`
- `uint16_t TxXferSize`
- `uint16_t TxXferCount`
- `uint8_t * pRxBuffPtr`
- `uint16_t RxXferSize`
- `uint16_t RxXferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `__IO HAL_SMARTCARD_StateTypeDef State`
- `__IO HAL_SMARTCARD_ErrorTypeDef ErrorCode`

##### Field Documentation

- `USART_TypeDef* SMARTCARD_HandleTypeDef::Instance`
- `SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init`
- `uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr`
- `uint16_t SMARTCARD_HandleTypeDef::TxXferSize`
- `uint16_t SMARTCARD_HandleTypeDef::TxXferCount`
- `uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr`
- `uint16_t SMARTCARD_HandleTypeDef::RxXferSize`
- `uint16_t SMARTCARD_HandleTypeDef::RxXferCount`
- `DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx`
- `HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock`
- `__IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State`
- `__IO HAL_SMARTCARD_ErrorTypeDef SMARTCARD_HandleTypeDef::ErrorCode`

#### 42.1.2 SMARTCARD\_InitTypeDef

*SMARTCARD\_InitTypeDef* is defined in the `stm32f4xx_hal_smartcard.h`

##### Data Fields

- `uint32_t BaudRate`
- `uint32_t WordLength`
- `uint32_t StopBits`
- `uint32_t Parity`
- `uint32_t Mode`
- `uint32_t CLKPolarity`
- `uint32_t CLKPhase`
- `uint32_t CLKLastBit`
- `uint32_t Prescaler`
- `uint32_t GuardTime`
- `uint32_t NACKState`

### Field Documentation

- **`uint32_t SMARTCARD_InitTypeDef::BaudRate`**
  - This member configures the SmartCard communication baud rate. The baud rate is computed using the following formula: IntegerDivider = ((PCLKx) / (8 \* (hirda->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32\_t) IntegerDivider) \* 8) + 0.5
- **`uint32_t SMARTCARD_InitTypeDef::WordLength`**
  - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [`SMARTCARD\_Word\_Length`](#)
- **`uint32_t SMARTCARD_InitTypeDef::StopBits`**
  - Specifies the number of stop bits transmitted. This parameter can be a value of [`SMARTCARD\_Stop\_Bits`](#)
- **`uint32_t SMARTCARD_InitTypeDef::Parity`**
  - Specifies the parity mode. This parameter can be a value of [`SMARTCARD\_Parity`](#)
- **`uint32_t SMARTCARD_InitTypeDef::Mode`**
  - Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [`SMARTCARD\_Mode`](#)
- **`uint32_t SMARTCARD_InitTypeDef::CLKPolarity`**
  - Specifies the steady state of the serial clock. This parameter can be a value of [`SMARTCARD\_Clock\_Polarity`](#)
- **`uint32_t SMARTCARD_InitTypeDef::CLKPhase`**
  - Specifies the clock transition on which the bit capture is made. This parameter can be a value of [`SMARTCARD\_Clock\_Phase`](#)
- **`uint32_t SMARTCARD_InitTypeDef::CLKLastBit`**
  - Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of [`SMARTCARD\_Last\_Bit`](#)
- **`uint32_t SMARTCARD_InitTypeDef::Prescaler`**
  - Specifies the SmartCard Prescaler. This parameter must be a number between Min\_Data = 0 and Max\_Data = 255
- **`uint32_t SMARTCARD_InitTypeDef::GuardTime`**
  - Specifies the SmartCard Guard Time. This parameter must be a number between Min\_Data = 0 and Max\_Data = 255
- **`uint32_t SMARTCARD_InitTypeDef::NACKState`**
  - Specifies the SmartCard NACK Transmission state. This parameter can be a value of [`SmartCard\_NACK\_State`](#)

### 42.1.3 USART\_TypeDef

**USART\_TypeDef** is defined in the stm32f439xx.h

#### Data Fields

- `__IO uint32_t SR`
- `__IO uint32_t DR`
- `__IO uint32_t BRR`
- `__IO uint32_t CR1`
- `__IO uint32_t CR2`
- `__IO uint32_t CR3`
- `__IO uint32_t GTPR`

#### Field Documentation

- `__IO uint32_t USART_TypeDef::SR`
  - USART Status register, Address offset: 0x00
- `__IO uint32_t USART_TypeDef::DR`
  - USART Data register, Address offset: 0x04
- `__IO uint32_t USART_TypeDef::BRR`
  - USART Baud rate register, Address offset: 0x08
- `__IO uint32_t USART_TypeDef::CR1`
  - USART Control register 1, Address offset: 0x0C
- `__IO uint32_t USART_TypeDef::CR2`
  - USART Control register 2, Address offset: 0x10
- `__IO uint32_t USART_TypeDef::CR3`
  - USART Control register 3, Address offset: 0x14
- `__IO uint32_t USART_TypeDef::GTPR`
  - USART Guard time and prescaler register, Address offset: 0x18

## 42.2 SMARTCARD Firmware driver API description

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

### 42.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD\_HandleTypeDef handle structure.
2. Initialize the SMARTCARD low level resources by implementing the HAL\_SMARTCARD\_MspInit() API:
  - a. Enable the USARTx interface clock.
  - b. SMARTCARD pins configuration:
    - Enable the clock for the SMARTCARD GPIOs.
    - Configure these SMARTCARD pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (HAL\_SMARTCARD\_Transmit\_IT() and HAL\_SMARTCARD\_Receive\_IT() APIs):
    - Configure the USARTx interrupt priority.
    - Enable the NVIC USART IRQ handle.

- d. DMA Configuration if you need to use DMA process (HAL\_SMARTCARD\_Transmit\_DMA() and HAL\_SMARTCARD\_Receive\_DMA() APIs):
  - Declare a DMA handle structure for the Tx/Rx stream.
  - Enable the DMAx interface clock.
  - Configure the declared DMA handle structure with the required Tx/Rx parameters.
  - Configure the DMA Tx/Rx Stream.
  - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
- 3. Program the Baud Rate, Word Length , Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the SMARTCARD Init structure.
- 4. Initialize the SMARTCARD registers by calling the HAL\_SMARTCARD\_Init() API:
  - These APIs configure also the low level Hardware GPIO, CLOCK, CORTEX...etc by calling the customized HAL\_SMARTCARD\_MspInit() API.



The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros \_\_SMARTCARD\_ENABLE\_IT() and \_\_SMARTCARD\_DISABLE\_IT() inside the transmit and receive process.

Three operation modes are available within this driver :

### **Polling mode IO operation**

- Send an amount of data in blocking mode using HAL\_SMARTCARD\_Transmit()
- Receive an amount of data in blocking mode using HAL\_SMARTCARD\_Receive()

### **Interrupt mode IO operation**

- Send an amount of data in non blocking mode using HAL\_SMARTCARD\_Transmit\_IT()
- At transmission end of transfer HAL\_SMARTCARD\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_SMARTCARD\_Receive\_IT()
- At reception end of transfer HAL\_SMARTCARD\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_RxCpltCallback
- In case of transfer Error, HAL\_SMARTCARD\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SMARTCARD\_ErrorCallback

### **DMA mode IO operation**

- Send an amount of data in non blocking mode (DMA) using HAL\_SMARTCARD\_Transmit\_DMA()

- At transmission end of transfer HAL\_SMARTCARD\_TxCpltCallback is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using  
HAL\_SMARTCARD\_Receive\_DMA()
- At reception end of transfer HAL\_SMARTCARD\_RxCpltCallback is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_RxCpltCallback
- In case of transfer Error, HAL\_SMARTCARD\_ErrorCallback() function is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_ErrorCallback

### SMARTCARD HAL driver macros list

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

- \_\_HAL\_SMARTCARD\_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

#### 42.2.2 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 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. Refer to the STM32F4xx reference manual (RM0090) for the SMARTCARD frame formats depending on the frame length defined by the M bit (8-bits or 9-bits).
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

- Prescaler
- GuardTime
- NACKState: The Smartcard NACK state
- Recommended SmartCard interface configuration to get the Answer to Reset from the Card:
  - Word Length = 9 Bits
  - 1.5 Stop Bit
  - Even parity
  - BaudRate = 12096 baud
  - Tx and Rx enabled

Please refer to the ISO 7816-3 specification for more details. -@- It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.

The HAL\_SMARTCARD\_Init() function follows the USART SmartCard configuration procedure (details for the procedure are available in reference manual (RM0329)).

- [\*\*HAL\\_SMARTCARD\\_Init\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_MspDelInit\(\)\*\*](#)

#### 42.2.3 IO operation functions

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 APIs return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_SMARTCARD\_TxCpltCallback(), HAL\_SMARTCARD\_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL\_SMARTCARD\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
  - [\*\*HAL\\_SMARTCARD\\_Transmit\(\)\*\*](#)
  - [\*\*HAL\\_SMARTCARD\\_Receive\(\)\*\*](#)
3. Non Blocking mode APIs with Interrupt are :
  - [\*\*HAL\\_SMARTCARD\\_Transmit\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_SMARTCARD\\_Receive\\_IT\(\)\*\*](#)
  - [\*\*HAL\\_SMARTCARD\\_IRQHandler\(\)\*\*](#)
4. Non Blocking mode functions with DMA are :
  - [\*\*HAL\\_SMARTCARD\\_Transmit\\_DMA\(\)\*\*](#)
  - [\*\*HAL\\_SMARTCARD\\_Receive\\_DMA\(\)\*\*](#)
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - [\*\*HAL\\_SMARTCARD\\_TxCpltCallback\(\)\*\*](#)
  - [\*\*HAL\\_SMARTCARD\\_RxCpltCallback\(\)\*\*](#)

- HAL\_SMARTCARD\_ErrorCallback()
- [\*\*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\(\)\*\*](#)

#### 42.2.4 Peripheral State and Errors functions

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

- 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\(\)\*\*](#)

#### 42.2.5 SmartCard Initialization and de-initialization functions

##### 42.2.5.1 HAL\_SMARTCARD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Init (</b> <a href="#"><b>SMARTCARD_HandleTypeDef * hsc</b></a> )
Function Description	Initializes the SmartCard mode according to the specified parameters in the SMARTCARD_InitTypeDef and create the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

##### 42.2.5.2 HAL\_SMARTCARD\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_DelInit (</b> <a href="#"><b>SMARTCARD_HandleTypeDef * hsc</b></a> )
---------------	--

---

Function Description	Deinitializes the USART SmartCard peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.2.5.3 HAL\_SMARTCARD\_MspInit

Function Name	<b>void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.2.5.4 HAL\_SMARTCARD\_MspDelInit

Function Name	<b>void HAL_SMARTCARD_MspDelInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 42.2.6 IO operation functions

### 42.2.6.1 HAL\_SMARTCARD\_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Transmit (</code> <code>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li><li>• <b>pData</b> : pointer to data buffer</li><li>• <b>Size</b> : amount of data to be sent</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 42.2.6.2 HAL\_SMARTCARD\_Receive

Function Name	<code>HAL_StatusTypeDef HAL_SMARTCARD_Receive (</code> <code>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</code> <code>Size, uint32_t Timeout)</code>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li><li>• <b>pData</b> : pointer to data buffer</li><li>• <b>Size</b> : amount of data to be received</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 42.2.6.3 HAL\_SMARTCARD\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (</b> <b>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</b> <b>Size)</b>
Function Description	Send an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> <li>• <b>pData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.2.6.4 HAL\_SMARTCARD\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (</b> <b>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</b> <b>Size)</b>
Function Description	Receive an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> <li>• <b>pData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.2.6.5 HAL\_SMARTCARD\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (</b> <b>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</b> <b>Size)</b>
Function Description	Send an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>

- **pData** : pointer to data buffer
  - **Size** : amount of data to be sent
- Return values
- **HAL status**
- Notes
- None.

#### 42.2.6.6 HAL\_SMARTCARD\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (</b> <b>SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t</b> <b>Size)</b>
Function Description	Receive an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> <li>• <b>pData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be received</li> </ul>
Return values	• <b>HAL status</b>
Notes	• When the SMARTCARD parity is enabled (PCE = 1) the data received contain the parity bit.s

#### 42.2.6.7 HAL\_SMARTCARD\_IRQHandler

Function Name	<b>void HAL_SMARTCARD_IRQHandler (</b> <b>SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	This function handles SMARTCARD interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.2.6.8 HAL\_SMARTCARD\_TxCpltCallback

Function Name	<b>void HAL_SMARTCARD_TxCpltCallback (</b> <b><i>SMARTCARD_HandleTypeDef</i> * hsc)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.2.6.9 HAL\_SMARTCARD\_RxCpltCallback

Function Name	<b>void HAL_SMARTCARD_RxCpltCallback (</b> <b><i>SMARTCARD_HandleTypeDef</i> * hsc)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.2.6.10 HAL\_SMARTCARD\_ErrorCallback

Function Name	<b>void HAL_SMARTCARD_ErrorCallback (</b> <b><i>SMARTCARD_HandleTypeDef</i> * hsc)</b>
Function Description	SMARTCARD error callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## Notes

- None.

### 42.2.7 Peripheral State and Errors functions

#### 42.2.7.1 HAL\_SMARTCARD\_GetState

Function Name	<b>HAL_SMARTCARD_StateTypeDef</b> <b>HAL_SMARTCARD_GetState ( SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	return the SMARTCARD state
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for SMARTCARD module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 42.2.7.2 HAL\_SMARTCARD\_GetError

Function Name	<b>uint32_t HAL_SMARTCARD_GetError ( SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	Return the SMARTCARD error code.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b> : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>SMARTCARD Error Code</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 42.3 SMARTCARD Firmware driver defines

#### 42.3.1 SMARTCARD

SMARTCARD

***SMARTCARD\_Clock\_Phase***

- #define: ***SMARTCARD\_PHASE\_1EDGE*** ((*uint32\_t*)0x00000000)
  
- #define: ***SMARTCARD\_PHASE\_2EDGE*** ((*uint32\_t*)*USART\_CR2\_CPHA*)

***SMARTCARD\_Clock\_Polarity***

- #define: ***SMARTCARD\_POLARITY\_LOW*** ((*uint32\_t*)0x00000000)
  
- #define: ***SMARTCARD\_POLARITY\_HIGH*** ((*uint32\_t*)*USART\_CR2\_CPOL*)

***SmartCard\_DMA\_Requests***

- #define: ***SMARTCARD\_DMAREQ\_TX*** ((*uint32\_t*)*USART\_CR3\_DMAT*)
  
- #define: ***SMARTCARD\_DMAREQ\_RX*** ((*uint32\_t*)*USART\_CR3\_DMAR*)

***SmartCard\_Flags***

- #define: ***SMARTCARD\_FLAG\_TXE*** ((*uint32\_t*)0x00000080)
  
- #define: ***SMARTCARD\_FLAG\_TC*** ((*uint32\_t*)0x00000040)
  
- #define: ***SMARTCARD\_FLAG\_RXNE*** ((*uint32\_t*)0x00000020)
  
- #define: ***SMARTCARD\_FLAG\_IDLE*** ((*uint32\_t*)0x00000010)
  
- #define: ***SMARTCARD\_FLAG\_ORE*** ((*uint32\_t*)0x00000008)

- #define: **SMARTCARD\_FLAG\_NE** ((*uint32\_t*)0x00000004)
- #define: **SMARTCARD\_FLAG\_FE** ((*uint32\_t*)0x00000002)
- #define: **SMARTCARD\_FLAG\_PE** ((*uint32\_t*)0x00000001)

#### *SmartCard Interrupt definition*

- #define: **SMARTCARD\_IT\_PE** ((*uint32\_t*)0x10000100)
- #define: **SMARTCARD\_IT\_TXE** ((*uint32\_t*)0x10000080)
- #define: **SMARTCARD\_IT\_TC** ((*uint32\_t*)0x10000040)
- #define: **SMARTCARD\_IT\_RXNE** ((*uint32\_t*)0x10000020)
- #define: **SMARTCARD\_IT\_IDLE** ((*uint32\_t*)0x10000010)
- #define: **SMARTCARD\_IT\_ERR** ((*uint32\_t*)0x20000001)

#### **SMARTCARD\_Last\_Bit**

- #define: **SMARTCARD\_LASTBIT\_DISABLE** ((*uint32\_t*)0x00000000)
- #define: **SMARTCARD\_LASTBIT\_ENABLE** ((*uint32\_t*)USART\_CR2\_LBCL)

***SMARTCARD\_Mode***

- #define: ***SMARTCARD\_MODE\_RX*** ((*uint32\_t*)*USART\_CR1\_RE*)
- #define: ***SMARTCARD\_MODE\_TX*** ((*uint32\_t*)*USART\_CR1\_TE*)
- #define: ***SMARTCARD\_MODE\_TX\_RX*** ((*uint32\_t*)(*USART\_CR1\_TE* |*USART\_CR1\_RE*))

***SmartCard\_NACK\_State***

- #define: ***SMARTCARD\_NACK\_ENABLED*** ((*uint32\_t*)*USART\_CR3\_NACK*)
- #define: ***SMARTCARD\_NACK\_DISABLED*** ((*uint32\_t*)0x00000000)

***SMARTCARD\_Parity***

- #define: ***SMARTCARD\_PARITY\_NONE*** ((*uint32\_t*)0x00000000)
- #define: ***SMARTCARD\_PARITY\_EVEN*** ((*uint32\_t*)*USART\_CR1\_PCE*)
- #define: ***SMARTCARD\_PARITY\_ODD*** ((*uint32\_t*)(*USART\_CR1\_PCE* |*USART\_CR1\_PS*))

***SMARTCARD\_Stop\_Bits***

- #define: ***SMARTCARD\_STOPBITS\_1*** ((*uint32\_t*)0x00000000)
- #define: ***SMARTCARD\_STOPBITS\_0\_5*** ((*uint32\_t*)*USART\_CR2\_STOP\_0*)
- #define: ***SMARTCARD\_STOPBITS\_2*** ((*uint32\_t*)*USART\_CR2\_STOP\_1*)

- #define: ***SMARTCARD\_STOPBITS\_1\_5*** ((*uint32\_t*)(***USART\_CR2\_STOP\_0 | USART\_CR2\_STOP\_1***))

***SMARTCARD\_Word\_Length***

- #define: ***SMARTCARD\_WORDLENGTH\_8B*** ((*uint32\_t*)0x00000000)
- #define: ***SMARTCARD\_WORDLENGTH\_9B*** ((*uint32\_t*)***USART\_CR1\_M***)

## 43 HAL SRAM Generic Driver

### 43.1 SRAM Firmware driver registers structures

#### 43.1.1 SRAM\_HandleTypeDef

*SRAM\_HandleTypeDef* is defined in the `stm32f4xx_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

### 43.2 SRAM Firmware driver API description

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

#### 43.2.1 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()

### 43.2.2 SRAM Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

- [\*HAL\\_SRAM\\_Init\(\)\*](#)
- [\*HAL\\_SRAM\\_DelInit\(\)\*](#)
- [\*HAL\\_SRAM\\_MspInit\(\)\*](#)
- [\*HAL\\_SRAM\\_MspDelInit\(\)\*](#)
- [\*HAL\\_SRAM\\_DMA\\_XferCpltCallback\(\)\*](#)
- [\*HAL\\_SRAM\\_DMA\\_XferErrorCallback\(\)\*](#)

### 43.2.3 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\(\)\*](#)

#### 43.2.4 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\(\)\*](#)

#### 43.2.5 SRAM State functions

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

- [\*HAL\\_SRAM\\_GetState\(\)\*](#)

#### 43.2.6 Initialization and de-initialization functions

##### 43.2.6.1 HAL\_SRAM\_Init

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Init ( <a href="#"><i>SRAM_HandleTypeDef</i></a> * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)</code>
Function Description	Performs the SRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>Timing</b> : Pointer to SRAM control timing structure</li> <li>• <b>ExtTiming</b> : Pointer to SRAM extended mode timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

##### 43.2.6.2 HAL\_SRAM\_Delinit

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Delinit ( <a href="#"><i>SRAM_HandleTypeDef</i></a> * hsram)</code>
Function Description	Performs the SRAM device De-initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

#### 43.2.6.3 HAL\_SRAM\_MspInit

Function Name	<b>void HAL_SRAM_MspInit ( <i>SRAM_HandleTypeDef</i> * hsram)</b>
Function Description	SRAM MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 43.2.6.4 HAL\_SRAM\_MspDelInit

Function Name	<b>void HAL_SRAM_MspDelInit ( <i>SRAM_HandleTypeDef</i> * hsram)</b>
Function Description	SRAM MSP DelInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 43.2.6.5 HAL\_SRAM\_DMA\_XferCpltCallback

Function Name	<b>void HAL_SRAM_DMA_XferCpltCallback ( <i>DMA_HandleTypeDef</i> * hdma)</b>
Function Description	DMA transfer complete callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li></ul>

---

Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

#### 43.2.6.6 HAL\_SRAM\_DMA\_XferErrorCallback

Function Name	<b>void HAL_SRAM_DMA_XferErrorCallback (</b> <b>DMA_HandleTypeDef * hdma)</b>
Function Description	DMA transfer complete error callback.
Parameters	<ul style="list-style-type: none"><li><b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>None.</li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

#### 43.2.7 Input and Output functions

##### 43.2.7.1 HAL\_SRAM\_Read\_8b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_8b (</b> <b>SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)</b>
Function Description	Reads 8-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"><li><b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li><li><b>pAddress</b> : Pointer to read start address</li><li><b>pDstBuffer</b> : Pointer to destination buffer</li><li><b>BufferSize</b> : Size of the buffer to read from memory</li></ul>
Return values	<ul style="list-style-type: none"><li><b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>None.</li></ul>

##### 43.2.7.2 HAL\_SRAM\_Write\_8b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Write_8b (</b> <b>SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)</b>
Function Description	Writes 8-bit buffer to SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b> : Pointer to write start address</li> <li>• <b>pSrcBuffer</b> : Pointer to source buffer to write</li> <li>• <b>BufferSize</b> : Size of the buffer to write to memory</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 43.2.7.3 HAL\_SRAM\_Read\_16b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_16b (</b> <b>SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)</b>
Function Description	Reads 16-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b> : Pointer to read start address</li> <li>• <b>pDstBuffer</b> : Pointer to destination buffer</li> <li>• <b>BufferSize</b> : Size of the buffer to read from memory</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 43.2.7.4 HAL\_SRAM\_Write\_16b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Write_16b (</b> <b>SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)</b>
Function Description	Writes 16-bit buffer to SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>

- **pAddress** : Pointer to write start address
  - **pSrcBuffer** : Pointer to source buffer to write
  - **BufferSize** : Size of the buffer to write to memory
  - **HAL status**
  - None.
- Return values
- Notes

#### 43.2.7.5 HAL\_SRAM\_Read\_32b

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Read_32b (</code> <code>SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t</code> <code>* pDstBuffer, uint32_t BufferSize)</code>
Function Description	Reads 32-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b> : Pointer to read start address</li> <li>• <b>pDstBuffer</b> : Pointer to destination buffer</li> <li>• <b>BufferSize</b> : Size of the buffer to read from memory</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 43.2.7.6 HAL\_SRAM\_Write\_32b

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Write_32b (</code> <code>SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t</code> <code>* pSrcBuffer, uint32_t BufferSize)</code>
Function Description	Writes 32-bit buffer to SRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b> : Pointer to write start address</li> <li>• <b>pSrcBuffer</b> : Pointer to source buffer to write</li> <li>• <b>BufferSize</b> : Size of the buffer to write to memory</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 43.2.7.7 HAL\_SRAM\_Read\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Read_DMA (</code> <code>SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t</code> <code>* pDstBuffer, uint32_t BufferSize)</code>
Function Description	Reads a Words data from the SRAM memory using DMA transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b> : Pointer to read start address</li> <li>• <b>pDstBuffer</b> : Pointer to destination buffer</li> <li>• <b>BufferSize</b> : Size of the buffer to read from memory</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 43.2.7.8 HAL\_SRAM\_Write\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Write_DMA (</code> <code>SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t</code> <code>* pSrcBuffer, uint32_t BufferSize)</code>
Function Description	Writes a Words data buffer to SRAM memory using DMA transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> <li>• <b>pAddress</b> : Pointer to write start address</li> <li>• <b>pSrcBuffer</b> : Pointer to source buffer to write</li> <li>• <b>BufferSize</b> : Size of the buffer to write to memory</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

## 43.2.8 Control functions

### 43.2.8.1 HAL\_SRAM\_WriteOperation\_Enable

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (</b> <b><i>SRAM_HandleTypeDef * hsram)</i></b>
Function Description	Enables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 43.2.8.2 HAL\_SRAM\_WriteOperation\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (</b> <b><i>SRAM_HandleTypeDef * hsram)</i></b>
Function Description	Disables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 43.2.9 State functions

##### 43.2.9.1 HAL\_SRAM\_GetState

Function Name	<b>HAL_SRAM_StateTypeDef HAL_SRAM_GetState (</b> <b><i>SRAM_HandleTypeDef * hsram)</i></b>
Function Description	Returns the SRAM controller state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b> : pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 43.3 SRAM Firmware driver defines

### 43.3.1 SRAM

SRAM

## 44 HAL SDRAM Generic Driver

### 44.1 SDRAM Firmware driver registers structures

#### 44.1.1 SDRAM\_HandleTypeDef

*SDRAM\_HandleTypeDef* is defined in the `stm32f4xx_hal_sdram.h`

##### Data Fields

- *FMC\_SDRAM\_TypeDef \* Instance*
- *FMC\_SDRAM\_InitTypeDef Init*
- *\_\_IO HAL\_SDRAM\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*
- *DMA\_HandleTypeDef \* hdma*

##### Field Documentation

- *FMC\_SDRAM\_TypeDef\* SDRAM\_HandleTypeDef::Instance*
  - Register base address
- *FMC\_SDRAM\_InitTypeDef SDRAM\_HandleTypeDef::Init*
  - SDRAM device configuration parameters
- *\_\_IO HAL\_SDRAM\_StateTypeDef SDRAM\_HandleTypeDef::State*
  - SDRAM access state
- *HAL\_LockTypeDef SDRAM\_HandleTypeDef::Lock*
  - SDRAM locking object
- *DMA\_HandleTypeDef\* SDRAM\_HandleTypeDef::hdma*
  - Pointer DMA handler

### 44.2 SDRAM Firmware driver API description

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

#### 44.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SDRAM memories. It uses the FMC layer functions to interface with SDRAM devices. The following sequence should be followed to configure the FMC to interface with SDRAM memories:

1. Declare a SDRAM\_HandleTypeDef handle structure, for example:  
`SDRAM_HandleTypeDef hdsram`
  - Fill the SDRAM\_HandleTypeDef handle "Init" field with the allowed values of the structure member.
  - Fill the SDRAM\_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SDRAM device
2. Declare a FMC\_SDRAM\_TimingTypeDef structure; for example:  
`FMC_SDRAM_TimingTypeDef Timing;` and fill its fields with the allowed values of the structure member.

3. Initialize the SDRAM Controller by calling the function `HAL_SDRAM_Init()`. This function performs the following sequence:
  - a. MSP hardware layer configuration using the function `HAL_SDRAM_MspInit()`
  - b. Control register configuration using the FMC SDRAM interface function `FMC_SDRAM_Init()`
  - c. Timing register configuration using the FMC SDRAM interface function `FMC_SDRAM_Timing_Init()`
  - d. Program the SDRAM external device by applying its initialization sequence according to the device plugged in your hardware. This step is mandatory for accessing the SDRAM device.
4. At this stage you can perform read/write accesses from/to the memory connected to the SDRAM Bank. You can perform either polling or DMA transfer using the following APIs:
  - `HAL_SDRAM_Read()`/`HAL_SDRAM_Write()` for polling read/write access
  - `HAL_SDRAM_Read_DMA()`/`HAL_SDRAM_Write_DMA()` for DMA read/write transfer
5. You can also control the SDRAM device by calling the control APIs `HAL_SDRAM_WriteOperation_Enable()`/`HAL_SDRAM_WriteOperation_Disable()` to respectively enable/disable the SDRAM write operation or the function `HAL_SDRAM_SendCommand()` to send a specified command to the SDRAM device. The command to be sent must be configured with the `FMC_SDRAM_CommandTypeDef` structure.
6. You can continuously monitor the SDRAM device HAL state by calling the function `HAL_SDRAM_GetState()`

#### 44.2.2 SDRAM Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the SDRAM memory

- `HAL_SDRAM_Init()`
- `HAL_SDRAM_DeInit()`
- `HAL_SDRAM_MspInit()`
- `HAL_SDRAM_MspDeInit()`
- `HAL_SDRAM_IRQHandler()`
- `HAL_SDRAM_RefreshErrorCallback()`
- `HAL_SDRAM_DMA_XferCpltCallback()`
- `HAL_SDRAM_DMA_XferErrorCallback()`

#### 44.2.3 SDRAM Input and Output functions

This section provides functions allowing to use and control the SDRAM memory

- `HAL_SDRAM_Read_8b()`
- `HAL_SDRAM_Write_8b()`
- `HAL_SDRAM_Read_16b()`
- `HAL_SDRAM_Write_16b()`
- `HAL_SDRAM_Read_32b()`
- `HAL_SDRAM_Write_32b()`
- `HAL_SDRAM_Read_DMA()`
- `HAL_SDRAM_Write_DMA()`

#### 44.2.4 SDRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SDRAM interface.

- `HAL_SDRAM_WriteProtection_Enable()`
- `HAL_SDRAM_WriteProtection_Disable()`
- `HAL_SDRAM_SendCommand()`
- `HAL_SDRAM_ProgramRefreshRate()`
- `HAL_SDRAM_SetAutoRefreshNumber()`
- `HAL_SDRAM_GetModeStatus()`

#### 44.2.5 SDRAM State functions

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

- `HAL_SDRAM_GetState()`

#### 44.2.6 Initialization and de-initialization functions

##### 44.2.6.1 HAL\_SDRAM\_Init

Function Name	<code>HAL_StatusTypeDef HAL_SDRAM_Init (</code> <code>SDRAM_HandleTypeDef * hsdran,</code> <code>FMC_SDRAM_TimingTypeDef * Timing)</code>
Function Description	Performs the SDRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdran</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li>• <b>Timing</b> : Pointer to SDRAM control timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

##### 44.2.6.2 HAL\_SDRAM\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_SDRAM_DelInit (</code> <code>SDRAM_HandleTypeDef * hsdran)</code>
Function Description	Perform the SDRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdran</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 44.2.6.3 HAL\_SDRAM\_MspInit

Function Name	<b>void HAL_SDRAM_MspInit ( <i>SDRAM_HandleTypeDef</i> * hsdram)</b>
Function Description	SDRAM MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 44.2.6.4 HAL\_SDRAM\_MspDeInit

Function Name	<b>void HAL_SDRAM_MspDeInit ( <i>SDRAM_HandleTypeDef</i> * hsdram)</b>
Function Description	SDRAM MSP DeInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 44.2.6.5 HAL\_SDRAM\_IRQHandler

Function Name	<b>void HAL_SDRAM_IRQHandler ( <i>SDRAM_HandleTypeDef</i> * hsdram)</b>
Function Description	This function handles SDRAM refresh error interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li></ul>

Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 44.2.6.6 HAL\_SDRAM\_RefreshErrorCallback

Function Name	<b>void HAL_SDRAM_RefreshErrorCallback (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram)</b>
Function Description	SDRAM Refresh error callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 44.2.6.7 HAL\_SDRAM\_DMA\_XferCpltCallback

Function Name	<b>void HAL_SDRAM_DMA_XferCpltCallback (</b> <b><i>DMA_HandleTypeDef</i> * hdma)</b>
Function Description	DMA transfer complete callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hdma</b> : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 44.2.6.8 HAL\_SDRAM\_DMA\_XferErrorCallback

Function Name	<b>void HAL_SDRAM_DMA_XferErrorCallback (</b> <b><i>DMA_HandleTypeDef</i> * hdma)</b>
---------------	--

Function Description	DMA transfer complete error callback.
Parameters	<ul style="list-style-type: none"> <li><b>hdma</b> : DMA handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 44.2.7 Input and Output functions

### 44.2.7.1 HAL\_SDRAM\_Read\_8b

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_Read_8b (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram, uint32_t * pAddress,</b> <b><i>uint8_t</i> * pDstBuffer, uint32_t BufferSize)</b>
Function Description	Reads 8-bit data buffer from the SDRAM memory.
Parameters	<ul style="list-style-type: none"> <li><b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li><b>pAddress</b> : Pointer to read start address</li> <li><b>pDstBuffer</b> : Pointer to destination buffer</li> <li><b>BufferSize</b> : Size of the buffer to read from memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 44.2.7.2 HAL\_SDRAM\_Write\_8b

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_Write_8b (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram, uint32_t * pAddress,</b> <b><i>uint8_t</i> * pSrcBuffer, uint32_t BufferSize)</b>
Function Description	Writes 8-bit data buffer to SDRAM memory.
Parameters	<ul style="list-style-type: none"> <li><b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li><b>pAddress</b> : Pointer to write start address</li> <li><b>pSrcBuffer</b> : Pointer to source buffer to write</li> <li><b>BufferSize</b> : Size of the buffer to write to memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 44.2.7.3 HAL\_SDRAM\_Read\_16b

Function Name	<code>HAL_StatusTypeDef HAL_SDRAM_Read_16b (     <b>SDRAM_HandleTypeDef</b> * hsdram, uint32_t * pAddress,     uint16_t * pDstBuffer, uint32_t BufferSize)</code>
Function Description	Reads 16-bit data buffer from the SDRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li>• <b>pAddress</b> : Pointer to read start address</li> <li>• <b>pDstBuffer</b> : Pointer to destination buffer</li> <li>• <b>BufferSize</b> : Size of the buffer to read from memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 44.2.7.4 HAL\_SDRAM\_Write\_16b

Function Name	<code>HAL_StatusTypeDef HAL_SDRAM_Write_16b (     <b>SDRAM_HandleTypeDef</b> * hsdram, uint32_t * pAddress,     uint16_t * pSrcBuffer, uint32_t BufferSize)</code>
Function Description	Writes 16-bit data buffer to SDRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li>• <b>pAddress</b> : Pointer to write start address</li> <li>• <b>pSrcBuffer</b> : Pointer to source buffer to write</li> <li>• <b>BufferSize</b> : Size of the buffer to write to memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 44.2.7.5 HAL\_SDRAM\_Read\_32b

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_Read_32b (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram, uint32_t * pAddress,</b> <b><i>uint32_t</i> * pDstBuffer, <i>uint32_t</i> BufferSize)</b>
Function Description	Reads 32-bit data buffer from the SDRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li>• <b>pAddress</b> : Pointer to read start address</li> <li>• <b>pDstBuffer</b> : Pointer to destination buffer</li> <li>• <b>BufferSize</b> : Size of the buffer to read from memory</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 44.2.7.6 HAL\_SDRAM\_Write\_32b

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_Write_32b (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram, uint32_t * pAddress,</b> <b><i>uint32_t</i> * pSrcBuffer, <i>uint32_t</i> BufferSize)</b>
Function Description	Writes 32-bit data buffer to SDRAM memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li>• <b>pAddress</b> : Pointer to write start address</li> <li>• <b>pSrcBuffer</b> : Pointer to source buffer to write</li> <li>• <b>BufferSize</b> : Size of the buffer to write to memory</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 44.2.7.7 HAL\_SDRAM\_Read\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_Read_DMA (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram, uint32_t * pAddress,</b> <b><i>uint32_t</i> * pDstBuffer, <i>uint32_t</i> BufferSize)</b>
Function Description	Reads a Words data from the SDRAM memory using DMA transfer.

Parameters	<ul style="list-style-type: none"> <li><b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li><b>pAddress</b> : Pointer to read start address</li> <li><b>pDstBuffer</b> : Pointer to destination buffer</li> <li><b>BufferSize</b> : Size of the buffer to read from memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 44.2.7.8 HAL\_SDRAM\_Write\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_Write_DMA (</b> <b>SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress,</b> <b>uint32_t * pSrcBuffer, uint32_t BufferSize)</b>
Function Description	Writes a Words data buffer to SDRAM memory using DMA transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li><b>pAddress</b> : Pointer to write start address</li> <li><b>pSrcBuffer</b> : Pointer to source buffer to write</li> <li><b>BufferSize</b> : Size of the buffer to write to memory</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 44.2.8 Control functions

##### 44.2.8.1 HAL\_SDRAM\_WriteProtection\_Enable

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Enable (</b> <b>SDRAM_HandleTypeDef * hsdram)</b>
Function Description	Enables dynamically SDRAM write protection.
Parameters	<ul style="list-style-type: none"> <li><b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 44.2.8.2 HAL\_SDRAM\_WriteProtection\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Disable (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram)</b>
Function Description	Disables dynamically SDRAM write protection.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 44.2.8.3 HAL\_SDRAM\_SendCommand

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_SendCommand (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram,</b> <b><i>FMC_SDRAM_CommandTypeDef</i> * Command, uint32_t</b> <b>Timeout)</b>
Function Description	Sends Command to the SDRAM bank.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li>• <b>Command</b> : SDRAM command structure</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 44.2.8.4 HAL\_SDRAM\_ProgramRefreshRate

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_ProgramRefreshRate (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram, uint32_t RefreshRate)</b>
Function Description	Programs the SDRAM Memory Refresh rate.

Parameters	<ul style="list-style-type: none"> <li><b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li><b>RefreshRate</b> : The SDRAM refresh rate value</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 44.2.8.5 HAL\_SDRAM\_SetAutoRefreshNumber

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_SetAutoRefreshNumber (</b> <b>SDRAM_HandleTypeDef * hsdram, uint32_t</b> <b>AutoRefreshNumber)</b>
Function Description	Sets the Number of consecutive SDRAM Memory auto Refresh commands.
Parameters	<ul style="list-style-type: none"> <li><b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li><b>AutoRefreshNumber</b> : The SDRAM auto Refresh number</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 44.2.8.6 HAL\_SDRAM\_GetModeStatus

Function Name	<b>uint32_t HAL_SDRAM_GetModeStatus (</b> <b>SDRAM_HandleTypeDef * hsdram)</b>
Function Description	Returns the SDRAM memory current mode.
Parameters	<ul style="list-style-type: none"> <li><b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>The SDRAM memory mode.</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 44.2.9 State functions

##### 44.2.9.1 HAL\_SDRAM\_GetState

Function Name	<b>HAL_SDRAM_StateTypeDef</b> <b>HAL_SDRAM_GetState (</b> <b><i>SDRAM_HandleTypeDef</i> * hsdram)</b>
Function Description	Returns the SDRAM state.
Parameters	<ul style="list-style-type: none"><li>• <b>hsdram</b> : pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 44.3 SDRAM Firmware driver defines

#### 44.3.1 SDRAM

SDRAM

## 45 HAL SPI Generic Driver

### 45.1 SPI Firmware driver registers structures

#### 45.1.1 SPI\_HandleTypeDef

*SPI\_HandleTypeDef* is defined in the `stm32f4xx_hal_spi.h`

##### Data Fields

- *SPI\_TypeDef \* Instance*
- *SPI\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *uint16\_t TxXferSize*
- *uint16\_t TxXferCount*
- *uint8\_t \* pRxBuffPtr*
- *uint16\_t RxXferSize*
- *uint16\_t RxXferCount*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *void(\* RxISR*
- *void(\* TxISR*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_SPI\_StateTypeDef State*
- *\_\_IO 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*
- *DMA\_HandleTypeDef\* SPI\_HandleTypeDef::hdmatx*
- *DMA\_HandleTypeDef\* SPI\_HandleTypeDef::hdmarx*
- *void(\* SPI\_HandleTypeDef::RxISR)(struct \_\_SPI\_HandleTypeDef \*hspi)*
- *void(\* SPI\_HandleTypeDef::TxISR)(struct \_\_SPI\_HandleTypeDef \*hspi)*
- *HAL\_LockTypeDef SPI\_HandleTypeDef::Lock*
- *\_\_IO HAL\_SPI\_StateTypeDef SPI\_HandleTypeDef::State*
- *\_\_IO HAL\_SPI\_ErrorTypeDef SPI\_HandleTypeDef::ErrorCode*

#### 45.1.2 SPI\_InitTypeDef

*SPI\_InitTypeDef* is defined in the `stm32f4xx_hal_spi.h`

---

## Data Fields

- *uint32\_t Mode*
- *uint32\_t Direction*
- *uint32\_t DataSize*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t NSS*
- *uint32\_t BaudRatePrescaler*
- *uint32\_t FirstBit*
- *uint32\_t TIMode*
- *uint32\_t CRCCalculation*
- *uint32\_t CRCPolynomial*

## Field Documentation

- *uint32\_t SPI\_InitTypeDef::Mode*
  - Specifies the SPI operating mode. This parameter can be a value of [SPI\\_mode](#)
- *uint32\_t SPI\_InitTypeDef::Direction*
  - Specifies the SPI Directional mode state. This parameter can be a value of [SPI\\_Direction\\_mode](#)
- *uint32\_t SPI\_InitTypeDef::DataSize*
  - Specifies the SPI data size. This parameter can be a value of [SPI\\_data\\_size](#)
- *uint32\_t SPI\_InitTypeDef::CLKPolarity*
  - Specifies the serial clock steady state. This parameter can be a value of [SPI\\_Clock\\_Polarity](#)
- *uint32\_t SPI\_InitTypeDef::CLKPhase*
  - Specifies the clock active edge for the bit capture. This parameter can be a value of [SPI\\_Clock\\_Phase](#)
- *uint32\_t SPI\_InitTypeDef::NSS*
  - Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of [SPI\\_Slave\\_Select\\_management](#)
- *uint32\_t SPI\_InitTypeDef::BaudRatePrescaler*
  - Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of [SPI\\_BaudRate\\_Prescaler](#)
- *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

### 45.1.3 SPI\_TypeDef

*SPI\_TypeDef* is defined in the stm32f439xx.h

#### Data Fields

- `__IO uint32_t CR1`
- `__IO uint32_t CR2`
- `__IO uint32_t SR`
- `__IO uint32_t DR`
- `__IO uint32_t CRCPR`
- `__IO uint32_t RXCRCR`
- `__IO uint32_t TXCRCR`
- `__IO uint32_t I2SCFGR`
- `__IO uint32_t I2SPR`

#### Field Documentation

- `__IO uint32_t SPI_TypeDef::CR1`
  - SPI control register 1 (not used in I2S mode), Address offset: 0x00
- `__IO uint32_t SPI_TypeDef::CR2`
  - SPI control register 2, Address offset: 0x04
- `__IO uint32_t SPI_TypeDef::SR`
  - SPI status register, Address offset: 0x08
- `__IO uint32_t SPI_TypeDef::DR`
  - SPI data register, Address offset: 0x0C
- `__IO uint32_t SPI_TypeDef::CRCPR`
  - SPI CRC polynomial register (not used in I2S mode), Address offset: 0x10
- `__IO uint32_t SPI_TypeDef::RXCRCR`
  - SPI RX CRC register (not used in I2S mode), Address offset: 0x14
- `__IO uint32_t SPI_TypeDef::TXCRCR`
  - SPI TX CRC register (not used in I2S mode), Address offset: 0x18
- `__IO uint32_t SPI_TypeDef::I2SCFGR`
  - SPI\_I2S configuration register, Address offset: 0x1C
- `__IO uint32_t SPI_TypeDef::I2SPR`
  - SPI\_I2S prescaler register, Address offset: 0x20

## 45.2 SPI Firmware driver API description

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

### 45.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI\_HandleTypeDef handle structure, for example: SPI\_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL\_SPI\_MspInit ()API:
  - a. Enable the SPIx interface clock

- b. SPI pins configuration
    - Enable the clock for the SPI GPIOs
    - Configure these SPI pins as alternate function push-pull
  - c. NVIC configuration if you need to use interrupt process
    - Configure the SPIx interrupt priority
    - Enable the NVIC SPI IRQ handle
  - d. DMA Configuration if you need to use DMA process
    - Declare a DMA\_HandleTypeDef handle structure for the transmit or receive stream
    - Enable the DMAx interface clock using
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx Stream
    - Associate the initialized hdma\_tx handle to the hspi DMA Tx or Rx handle
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream
3. Program the Mode, Direction , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
  4. Initialize the SPI registers by calling the HAL\_SPI\_Init() API:
    - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc by calling the customized HAL\_SPI\_MspInit() API.

#### 45.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL\_SPI\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function HAL\_SPI\_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
- Call the function HAL\_SPI\_DeInit() to restore the default configuration of the selected SPIx peripheral.
- ***HAL\_SPI\_Init()***
- ***HAL\_SPI\_DeInit()***
- ***HAL\_SPI\_MspInit()***
- ***HAL\_SPI\_MspDeInit()***

#### 45.2.3 IO operation functions

The SPI supports master and slave mode :

1. There are two modes of transfer:

- Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_SPI\_TxCpltCallback(), HAL\_SPI\_RxCpltCallback() and HAL\_SPI\_TxRxCPtCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL\_SPI\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs 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 non Blocking mode:
    - HAL\_SPI\_TxCpltCallback()
    - HAL\_SPI\_RxCpltCallback()
    - HAL\_SPI\_ErrorCallback()
    - HAL\_SPI\_TxRxCPtCallback()
    - ***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\_TxRxCPtCallback()***
    - ***HAL\_SPI\_ErrorCallback()***

#### 45.2.4 Peripheral State and Errors functions

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

- HAL\_SPI\_GetState() API can be helpful to check in run-time the state of the SPI peripheral
- HAL\_SPI\_GetError() check in run-time Errors occurring during communication
- ***HAL\_SPI\_GetState()***
- ***HAL\_SPI\_GetError()***

### 45.2.5 Initialization and de-initialization functions

#### 45.2.5.1 HAL\_SPI\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Init ( <i>SPI_HandleTypeDef</i> * <i>hspi</i>)</b>
Function Description	Initializes the SPI according to the specified parameters in the <i>SPI_InitTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b> : pointer to a <i>SPI_HandleTypeDef</i> structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 45.2.5.2 HAL\_SPI\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DelInit ( <i>SPI_HandleTypeDef</i> * <i>hspi</i>)</b>
Function Description	Deinitializes the SPI peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b> : pointer to a <i>SPI_HandleTypeDef</i> structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 45.2.5.3 HAL\_SPI\_MspInit

Function Name	<b>void HAL_SPI_MspInit ( <i>SPI_HandleTypeDef</i> * <i>hspi</i>)</b>
Function Description	SPI MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b> : pointer to a <i>SPI_HandleTypeDef</i> structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 45.2.5.4 HAL\_SPI\_MspDelInit

Function Name	<b>void HAL_SPI_MspDelInit ( <i>SPI_HandleTypeDef</i> * hspi)</b>
Function Description	SPI MSP DelInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 45.2.6 IO operation functions

##### 45.2.6.1 HAL\_SPI\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit ( <i>SPI_HandleTypeDef</i> * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be sent</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

##### 45.2.6.2 HAL\_SPI\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive ( <i>SPI_HandleTypeDef</i> * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
---------------	---

Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be sent</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 45.2.6.3 HAL\_SPI\_TransmitReceive

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pTxData</b> : pointer to transmission data buffer</li> <li>• <b>pRxData</b> : pointer to reception data buffer to be</li> <li>• <b>Size</b> : amount of data to be sent</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 45.2.6.4 HAL\_SPI\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_IT (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

**45.2.6.5 HAL\_SPI\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_IT (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**45.2.6.6 HAL\_SPI\_TransmitReceive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
Function Description	Transmit and Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pTxData</b> : pointer to transmission data buffer</li> <li>• <b>pRxData</b> : pointer to reception data buffer to be</li> <li>• <b>Size</b> : amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**45.2.6.7 HAL\_SPI\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_DMA (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmit an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b> : pointer to data buffer</li> <li>• <b>Size</b> : amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 45.2.6.8 HAL\_SPI\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_DMA (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pData</b> : pointer to data buffer</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• <b>Size</b> : amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the CRC feature is enabled the pData Length must be Size + 1.</li> </ul>

#### 45.2.6.9 HAL\_SPI\_TransmitReceive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (</b> <b>SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
Function Description	Transmit and Receive an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li>• <b>pTxData</b> : pointer to transmission data buffer</li> </ul>

---

Parameters	<ul style="list-style-type: none"> <li><b>pRxData</b> : pointer to reception data buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>Size</b> : amount of data to be sent</li> </ul>
Notes	<ul style="list-style-type: none"> <li><b>HAL status</b></li> <li>When the CRC feature is enabled the pRxData Length must be Size + 1</li> </ul>

#### 45.2.6.10 HAL\_SPI\_IRQHandler

Function Name	<b>void HAL_SPI_IRQHandler ( <i>SPI_HandleTypeDef</i> * hspi)</b>
Function Description	This function handles SPI interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 45.2.6.11 HAL\_SPI\_TxCpltCallback

Function Name	<b>void HAL_SPI_TxCpltCallback ( <i>SPI_HandleTypeDef</i> * hspi)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li><b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

#### 45.2.6.12 HAL\_SPI\_RxCpltCallback

Function Name	<b>void HAL_SPI_RxCpltCallback ( <i>SPI_HandleTypeDef</i> * hspi)</b>
---------------	---

Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 45.2.6.13 HAL\_SPI\_TxRxCpltCallback

Function Name	<b>void HAL_SPI_TxRxCpltCallback ( SPI_HandleTypeDef * hspi)</b>
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 45.2.6.14 HAL\_SPI\_ErrorCallback

Function Name	<b>void HAL_SPI_ErrorCallback ( SPI_HandleTypeDef * hspi)</b>
Function Description	SPI error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 45.2.7 Peripheral State and Errors functions

#### 45.2.7.1 HAL\_SPI\_GetState

---

Function Name	<b>HAL_SPI_StateTypeDef HAL_SPI_GetState (</b> <i>SPI_HandleTypeDef * hspi)</i>
Function Description	Return the SPI state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 45.2.7.2 HAL\_SPI\_GetError

Function Name	<b>HAL_SPI_ErrorTypeDef HAL_SPI_GetError (</b> <i>SPI_HandleTypeDef * hspi)</i>
Function Description	Return the SPI error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b> : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>SPI Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 45.3 SPI Firmware driver defines

### 45.3.1 SPI

SPI

*SPI\_BaudRate\_Prescaler*

- #define: **SPI\_BAUDRATEPRESCALER\_2 ((uint32\_t)0x00000000)**
- #define: **SPI\_BAUDRATEPRESCALER\_4 ((uint32\_t)0x00000008)**
- #define: **SPI\_BAUDRATEPRESCALER\_8 ((uint32\_t)0x00000010)**

- #define: **SPI\_BAUDRATEPRESCALER\_16** ((*uint32\_t*)0x00000018)
- #define: **SPI\_BAUDRATEPRESCALER\_32** ((*uint32\_t*)0x00000020)
- #define: **SPI\_BAUDRATEPRESCALER\_64** ((*uint32\_t*)0x00000028)
- #define: **SPI\_BAUDRATEPRESCALER\_128** ((*uint32\_t*)0x00000030)
- #define: **SPI\_BAUDRATEPRESCALER\_256** ((*uint32\_t*)0x00000038)

#### ***SPI\_Clock\_Phase***

- #define: **SPI\_PHASE\_1EDGE** ((*uint32\_t*)0x00000000)
- #define: **SPI\_PHASE\_2EDGE SPI\_CR1\_CPHA**

#### ***SPI\_Clock\_Polarity***

- #define: **SPI\_POLARITY\_LOW** ((*uint32\_t*)0x00000000)
- #define: **SPI\_POLARITY\_HIGH SPI\_CR1\_CPOL**

#### ***SPI\_CRC\_Calculation***

- #define: **SPI\_CRCCALCULATION\_DISABLED** ((*uint32\_t*)0x00000000)
- #define: **SPI\_CRCCALCULATION\_ENABLED SPI\_CR1\_CRCEN**

***SPI\_data\_size***

- #define: ***SPI\_DATASIZE\_8BIT*** ((*uint32\_t*)0x00000000)
  
- #define: ***SPI\_DATASIZE\_16BIT SPI\_CR1\_DFF***

***SPI\_Direction\_mode***

- #define: ***SPI\_DIRECTION\_2LINES*** ((*uint32\_t*)0x00000000)
  
- #define: ***SPI\_DIRECTION\_2LINES\_RXONLY SPI\_CR1\_RXONLY***
  
- #define: ***SPI\_DIRECTION\_1LINE SPI\_CR1\_BIDIMODE***

***SPI\_Flag\_definition***

- #define: ***SPI\_FLAG\_RXNE SPI\_SR\_RXNE***

- #define: ***SPI\_FLAG\_TXE SPI\_SR\_TXE***

- #define: ***SPI\_FLAG\_CRCERR SPI\_SR\_CRCERR***

- #define: ***SPI\_FLAG\_MODF SPI\_SR\_MODF***

- #define: ***SPI\_FLAG\_OVR SPI\_SR\_OVR***

- #define: ***SPI\_FLAG\_BSY SPI\_SR\_BSY***

- #define: **SPI\_FLAG\_FRE SPI\_SR\_FRE**

***SPI\_Interrupt\_configuration\_definition***

- #define: **SPI\_IT\_TXE SPI\_CR2\_TXEIE**
- #define: **SPI\_IT\_RXNE SPI\_CR2\_RXNEIE**
- #define: **SPI\_IT\_ERR SPI\_CR2\_ERRIE**

***SPI\_mode***

- #define: **SPI\_MODE\_SLAVE ((uint32\_t)0x00000000)**
- #define: **SPI\_MODE\_MASTER (SPI\_CR1\_MSTR | SPI\_CR1\_SSI)**

***SPI\_MSB\_LSB\_transmission***

- #define: **SPI\_FIRSTBIT\_MSB ((uint32\_t)0x00000000)**
- #define: **SPI\_FIRSTBIT\_LSB SPI\_CR1\_LSBFIRST**

***SPI\_Slave\_Select\_management***

- #define: **SPI\_NSS\_SOFT SPI\_CR1\_SSM**
- #define: **SPI\_NSS\_HARD\_INPUT ((uint32\_t)0x00000000)**
- #define: **SPI\_NSS\_HARD\_OUTPUT ((uint32\_t)0x00040000)**

***SPI\_TI\_mode***

- #define: ***SPI\_TIMODE\_DISABLED ((uint32\_t)0x00000000)***
  
- #define: ***SPI\_TIMODE\_ENABLED SPI\_CR2\_FRF***

## 46 HAL TIM Generic Driver

### 46.1 TIM Firmware driver registers structures

#### 46.1.1 TIM\_HandleTypeDef

*TIM\_HandleTypeDef* is defined in the `stm32f4xx_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 DMA\_Handle\_index
- *HAL\_LockTypeDef TIM\_HandleTypeDef::Lock*
  - Locking object
- *\_\_IO HAL\_TIM\_StateTypeDef TIM\_HandleTypeDef::State*
  - TIM operation state

#### 46.1.2 TIM\_Base\_InitTypeDef

*TIM\_Base\_InitTypeDef* is defined in the `stm32f4xx_hal_tim.h`

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t CounterMode*
- *uint32\_t Period*
- *uint32\_t ClockDivision*
- *uint32\_t RepetitionCounter*

##### Field Documentation

- ***uint32\_t TIM\_Base\_InitTypeDef::Prescaler***
  - Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t TIM\_Base\_InitTypeDef::CounterMode***
  - Specifies the counter mode. This parameter can be a value of [TIM\\_Counter\\_Mode](#)
- ***uint32\_t TIM\_Base\_InitTypeDef::Period***
  - Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- ***uint32\_t TIM\_Base\_InitTypeDef::ClockDivision***
  - Specifies the clock division. This parameter can be a value of [TIM\\_ClockDivision](#)
- ***uint32\_t TIM\_Base\_InitTypeDef::RepetitionCounter***
  - Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to: the number of PWM periods in edge-aligned mode the number of half PWM period in center-aligned mode This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF. This parameter is valid only for TIM1 and TIM8.

#### 46.1.3 TIM\_OC\_InitTypeDef

**TIM\_OC\_InitTypeDef** is defined in the `stm32f4xx_hal_tim.h`

##### Data Fields

- ***uint32\_t OCMode***
- ***uint32\_t Pulse***
- ***uint32\_t OCPolarity***
- ***uint32\_t OCNPolarity***
- ***uint32\_t OCFastMode***
- ***uint32\_t OCIdleState***
- ***uint32\_t OCNIdleState***

##### Field Documentation

- ***uint32\_t TIM\_OC\_InitTypeDef::OCMode***
  - Specifies the TIM mode. This parameter can be a value of [TIM\\_Output\\_Compare\\_and\\_PWM\\_modes](#)
- ***uint32\_t TIM\_OC\_InitTypeDef::Pulse***
  - Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t TIM\_OC\_InitTypeDef::OCPolarity***
  - Specifies the output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_Polarity](#)
- ***uint32\_t TIM\_OC\_InitTypeDef::OCNPolarity***
  - Specifies the complementary output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Polarity](#)

- ***uint32\_t TIM\_OC\_InitTypeDef::OCFastMode***
  - Specifies the Fast mode state. This parameter can be a value of ***TIM\_Output\_Fast\_State***
- ***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***
- ***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***

#### 46.1.4 **TIM\_IC\_InitTypeDef**

***TIM\_IC\_InitTypeDef*** is defined in the `stm32f4xx_hal_tim.h`

##### Data Fields

- ***uint32\_t IC\_Polarity***
- ***uint32\_t IC\_Selection***
- ***uint32\_t IC\_Prescaler***
- ***uint32\_t IC\_Filter***

##### Field Documentation

- ***uint32\_t TIM\_IC\_InitTypeDef::IC\_Polarity***
  - Specifies the active edge of the input signal. This parameter can be a value of ***TIM\_Input\_Capture\_Polarity***
- ***uint32\_t TIM\_IC\_InitTypeDef::IC\_Selection***
  - Specifies the input. This parameter can be a value of ***TIM\_Input\_Capture\_Selection***
- ***uint32\_t TIM\_IC\_InitTypeDef::IC\_Prescaler***
  - Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM\_Input\_Capture\_Prescaler***
- ***uint32\_t TIM\_IC\_InitTypeDef::IC\_Filter***
  - Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

#### 46.1.5 **TIM\_OnePulse\_InitTypeDef**

***TIM\_OnePulse\_InitTypeDef*** is defined in the `stm32f4xx_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 IC\_Polarity***

- *uint32\_t ICSelection*
- *uint32\_t ICFfilter*

#### Field Documentation

- *uint32\_t TIM\_OnePulse\_InitTypeDef::OCMode*
  - Specifies the TIM mode. This parameter can be a value of [TIM\\_Output\\_Compare\\_and\\_PWM\\_modes](#)
- *uint32\_t TIM\_OnePulse\_InitTypeDef::Pulse*
  - Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- *uint32\_t TIM\_OnePulse\_InitTypeDef::OCPolarity*
  - Specifies the output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_Polarity](#)
- *uint32\_t TIM\_OnePulse\_InitTypeDef::OCNPolarity*
  - Specifies the complementary output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Polarity](#)
- *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](#)
- *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](#)
- *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

#### 46.1.6 TIM\_ClockConfigTypeDef

*TIM\_ClockConfigTypeDef* is defined in the `stm32f4xx_hal_tim.h`

##### Data Fields

- *uint32\_t ClockSource*
- *uint32\_t ClockPolarity*
- *uint32\_t ClockPrescaler*
- *uint32\_t ClockFilter*

#### Field Documentation

- *uint32\_t TIM\_ClockConfigTypeDef::ClockSource*

- TIM clock sources. This parameter can be a value of [\*\*TIM\\_Clock\\_Source\*\*](#)
- ***uint32\_t TIM\_ClockConfigTypeDef::ClockPolarity***
  - TIM clock polarity. This parameter can be a value of [\*\*TIM\\_Clock\\_Polarity\*\*](#)
- ***uint32\_t TIM\_ClockConfigTypeDef::ClockPrescaler***
  - TIM clock prescaler. This parameter can be a value of [\*\*TIM\\_Clock\\_Prescaler\*\*](#)
- ***uint32\_t TIM\_ClockConfigTypeDef::ClockFilter***
  - TIM clock filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

#### 46.1.7 **TIM\_ClearInputConfigTypeDef**

***TIM\_ClearInputConfigTypeDef*** is defined in the `stm32f4xx_hal_tim.h`

##### Data Fields

- ***uint32\_t ClearInputState***
- ***uint32\_t ClearInputSource***
- ***uint32\_t ClearInputPolarity***
- ***uint32\_t ClearInputPrescaler***
- ***uint32\_t ClearInputFilter***

##### Field Documentation

- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputState***
  - TIM clear Input state. This parameter can be ENABLE or DISABLE
- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputSource***
  - TIM clear Input sources. This parameter can be a value of [\*\*TIM\\_ClearInput\\_Source\*\*](#)
- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputPolarity***
  - TIM Clear Input polarity. This parameter can be a value of [\*\*TIM\\_ClearInput\\_Polarity\*\*](#)
- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputPrescaler***
  - TIM Clear Input prescaler. This parameter can be a value of [\*\*TIM\\_ClearInput\\_Prescaler\*\*](#)
- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputFilter***
  - TIM Clear Input filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

#### 46.1.8 **TIM\_SlaveConfigTypeDef**

***TIM\_SlaveConfigTypeDef*** is defined in the `stm32f4xx_hal_tim.h`

##### Data Fields

- ***uint32\_t SlaveMode***
- ***uint32\_t InputTrigger***
- ***uint32\_t TriggerPolarity***
- ***uint32\_t TriggerPrescaler***
- ***uint32\_t TriggerFilter***

## Field Documentation

- `uint32_t TIM_SlaveConfigTypeDef::SlaveMode`
  - Slave mode selection. This parameter can be a value of [TIM\\_Slave\\_Mode](#)
- `uint32_t TIM_SlaveConfigTypeDef::InputTrigger`
  - Input Trigger source. This parameter can be a value of [TIM\\_Trigger\\_Selection](#)
- `uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity`
  - Input Trigger polarity. This parameter can be a value of [TIM\\_Trigger\\_Polarity](#)
- `uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler`
  - Input trigger prescaler. This parameter can be a value of [TIM\\_Trigger\\_Prescaler](#)
- `uint32_t TIM_SlaveConfigTypeDef::TriggerFilter`
  - Input trigger filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

### 46.1.9 TIM\_Encoder\_InitTypeDef

`TIM_Encoder_InitTypeDef` is defined in the `stm32f4xx_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

#### 46.1.10 TIM\_TypeDef

`TIM_TypeDef` is defined in the `stm32f439xx.h`

##### Data Fields

- `_IO uint32_t CR1`
- `_IO uint32_t CR2`
- `_IO uint32_t SMCR`
- `_IO uint32_t DIER`
- `_IO uint32_t SR`
- `_IO uint32_t EGR`
- `_IO uint32_t CCMR1`
- `_IO uint32_t CCMR2`
- `_IO uint32_t CCER`
- `_IO uint32_t CNT`
- `_IO uint32_t PSC`
- `_IO uint32_t ARR`
- `_IO uint32_t RCR`
- `_IO uint32_t CCR1`
- `_IO uint32_t CCR2`
- `_IO uint32_t CCR3`
- `_IO uint32_t CCR4`
- `_IO uint32_t BDTR`
- `_IO uint32_t DCR`
- `_IO uint32_t DMAR`
- `_IO uint32_t OR`

##### Field Documentation

- `_IO uint32_t TIM_TypeDef::CR1`
  - TIM control register 1, Address offset: 0x00
- `_IO uint32_t TIM_TypeDef::CR2`
  - TIM control register 2, Address offset: 0x04
- `_IO uint32_t TIM_TypeDef::SMCR`
  - TIM slave mode control register, Address offset: 0x08
- `_IO uint32_t TIM_TypeDef::DIER`

- TIM DMA/interrupt enable register, Address offset: 0x0C
- **`__IO uint32_t TIM_TypeDef::SR`**
  - TIM status register, Address offset: 0x10
- **`__IO uint32_t TIM_TypeDef::EGR`**
  - TIM event generation register, Address offset: 0x14
- **`__IO uint32_t TIM_TypeDef::CCMR1`**
  - TIM capture/compare mode register 1, Address offset: 0x18
- **`__IO uint32_t TIM_TypeDef::CCMR2`**
  - TIM capture/compare mode register 2, Address offset: 0x1C
- **`__IO uint32_t TIM_TypeDef::CCER`**
  - TIM capture/compare enable register, Address offset: 0x20
- **`__IO uint32_t TIM_TypeDef::CNT`**
  - TIM counter register, Address offset: 0x24
- **`__IO uint32_t TIM_TypeDef::PSC`**
  - TIM prescaler, Address offset: 0x28
- **`__IO uint32_t TIM_TypeDef::ARR`**
  - TIM auto-reload register, Address offset: 0x2C
- **`__IO uint32_t TIM_TypeDef::RCR`**
  - TIM repetition counter register, Address offset: 0x30
- **`__IO uint32_t TIM_TypeDef::CCR1`**
  - TIM capture/compare register 1, Address offset: 0x34
- **`__IO uint32_t TIM_TypeDef::CCR2`**
  - TIM capture/compare register 2, Address offset: 0x38
- **`__IO uint32_t TIM_TypeDef::CCR3`**
  - TIM capture/compare register 3, Address offset: 0x3C
- **`__IO uint32_t TIM_TypeDef::CCR4`**
  - TIM capture/compare register 4, Address offset: 0x40
- **`__IO uint32_t TIM_TypeDef::BDTR`**
  - TIM break and dead-time register, Address offset: 0x44
- **`__IO uint32_t TIM_TypeDef::DCR`**
  - TIM DMA control register, Address offset: 0x48
- **`__IO uint32_t TIM_TypeDef::DMAR`**
  - TIM DMA address for full transfer, Address offset: 0x4C
- **`__IO uint32_t TIM_TypeDef::OR`**
  - TIM option register, Address offset: 0x50

## 46.2 TIM Firmware driver API description

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

### 46.2.1 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

### 46.2.2 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()

### 46.2.3 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\_DelInit\(\)`](#)
- [`HAL\_TIM\_Base\_MspInit\(\)`](#)
- [`HAL\_TIM\_Base\_MspDelInit\(\)`](#)
- [`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\(\)`](#)

### 46.2.4 Peripheral State functions

This subsection permits 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\(\)`](#)

### 46.2.5 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\_DelInit\(\)`](#)
- [`HAL\_TIM\_OC\_MspInit\(\)`](#)
- [`HAL\_TIM\_OC\_MspDelInit\(\)`](#)
- [`HAL\_TIM\_OC\_Start\(\)`](#)

- [\*HAL\\_TIM\\_OC\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Stop\\_DMA\(\)\*](#)

#### 46.2.6 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\(\)\*](#)

#### 46.2.7 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\(\)\*](#)

## 46.2.8 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\\_DelInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_MspDelInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Stop\\_IT\(\)\*](#)

## 46.2.9 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\\_DelInit\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_MspDelInit\(\)\*](#)
- [\*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\(\)\*](#)

## 46.2.10 IRQ handler management

This section provides Timer IRQ handler function.

- [\*HAL\\_TIM\\_IRQHandler\(\)\*](#)

### 46.2.11 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\\_ReadCapturedValue\(\)\*](#)

### 46.2.12 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\(\)\*](#)

### 46.2.13 Time Base functions

#### 46.2.13.1 HAL\_TIM\_Base\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Init (</b> <b><i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.

---

Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.13.2 HAL\_TIM\_Base\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_DeInit (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	Deinitializes the TIM Base peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.13.3 HAL\_TIM\_Base\_MspInit

Function Name	<b>void HAL_TIM_Base_MspInit (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	Initializes the TIM Base MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.13.4 HAL\_TIM\_Base\_MspDeInit

Function Name	<b>void HAL_TIM_Base_MspDeInit (</b> <i>TIM_HandleTypeDef * htim)</i>
---------------	---

Function Description	Deinitializes TIM Base MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.13.5 HAL\_TIM\_Base\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start (</b> <b>TIM_HandleTypeDef * htim)</b>
Function Description	Starts the TIM Base generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.13.6 HAL\_TIM\_Base\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop (</b> <b>TIM_HandleTypeDef * htim)</b>
Function Description	Stops the TIM Base generation.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.13.7 HAL\_TIM\_Base\_Start\_IT

---

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start_IT (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Starts the TIM Base generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.13.8 HAL\_TIM\_Base\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Stops the TIM Base generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.13.9 HAL\_TIM\_Base\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length</i></b> )
Function Description	Starts the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>pData</b> : The source Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from memory to peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 46.2.13.10 HAL\_TIM\_Base\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Stops the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 46.2.14 Peripheral State functions

### 46.2.14.1 HAL\_TIM\_Base\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Return the TIM Base state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 46.2.14.2 HAL\_TIM\_OC\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Return the TIM OC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL state</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.14.3 HAL\_TIM\_PWM\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Return the TIM PWM state.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.14.4 HAL\_TIM\_IC\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Return the TIM Input Capture state.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.14.5 HAL\_TIM\_OnePulse\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Return the TIM One Pulse Mode state.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>

---

Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.14.6 HAL\_TIM\_Encoder\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	Return the TIM Encoder Mode state.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.15 Time Output Compare functions

##### 46.2.15.1 HAL\_TIM\_OC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Init (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

##### 46.2.15.2 HAL\_TIM\_OC\_DelInit

---

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_DelInit (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.15.3 HAL\_TIM\_OC\_MspInit

Function Name	<b>void HAL_TIM_OC_MspInit (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	Initializes the TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.15.4 HAL\_TIM\_OC\_MspDelInit

Function Name	<b>void HAL_TIM_OC_MspDelInit (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	Deinitializes TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.15.5 HAL\_TIM\_OC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>uint32_t</i> <i>Channel</i>)</b>
Function Description	Starts the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.15.6 HAL\_TIM\_OC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>uint32_t</i> <i>Channel</i>)</b>
Function Description	Stops the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.15.7 HAL\_TIM\_OC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_IT ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>uint32_t</i> <i>Channel</i>)</b>
Function Description	Starts the TIM Output Compare signal generation in interrupt

---

	mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> : TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> : TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> : TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.15.8 HAL\_TIM\_OC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> : TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> : TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> : TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.15.9 HAL\_TIM\_OC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM Output Compare signal generation in DMA mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <b>TIM_CHANNEL_1</b> : TIM Channel 1 selected</li> <li>- <b>TIM_CHANNEL_2</b> : TIM Channel 2 selected</li> <li>- <b>TIM_CHANNEL_3</b> : TIM Channel 3 selected</li> <li>- <b>TIM_CHANNEL_4</b> : TIM Channel 4 selected</li> </ul> </li> <li>• <b>pData</b> : The source Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from memory to TIM peripheral</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 46.2.15.10 HAL\_TIM\_OC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <b>TIM_CHANNEL_1</b> : TIM Channel 1 selected</li> <li>- <b>TIM_CHANNEL_2</b> : TIM Channel 2 selected</li> <li>- <b>TIM_CHANNEL_3</b> : TIM Channel 3 selected</li> <li>- <b>TIM_CHANNEL_4</b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 46.2.16 Time PWM functions

#### 46.2.16.1 HAL\_TIM\_PWM\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Init (</b> <b><i>TIM_HandleTypeDef</i> * htim)</b>
---------------	--

Function Description	Initializes the TIM PWM Time Base according to the specified parameters in the <code>TIM_HandleTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.16.2 HAL\_TIM\_PWM\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_DeInit (</code> <code>TIM_HandleTypeDef * htim)</code>
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.16.3 HAL\_TIM\_PWM\_MspInit

Function Name	<code>void HAL_TIM_PWM_MspInit ( TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM PWM MSP.
Parameters	<ul style="list-style-type: none"> <li>• <code>htim</code> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.16.4 HAL\_TIM\_PWM\_MspDeInit

Function Name	<b>void HAL_TIM_PWM_MspDeInit ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Deinitializes TIM PWM MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.16.5 HAL\_TIM\_PWM\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start ( <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Starts the PWM signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.16.6 HAL\_TIM\_PWM\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop ( <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.16.7 HAL\_TIM\_PWM\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.16.8 HAL\_TIM\_PWM\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

**46.2.16.9 HAL\_TIM\_PWM\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel, uint32_t *</b> <b><i>pData, uint16_t Length</i>)</b>
Function Description	Starts the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> <li>• <b>pData</b> : The source Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from memory to TIM peripheral</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**46.2.16.10 HAL\_TIM\_PWM\_Stop\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

- 
- |       |   |
|-------|---|
| Notes | <ul style="list-style-type: none"> <li>None.</li> </ul> |
|-------|---|

## 46.2.17 Time Input Capture functions

### 46.2.17.1 HAL\_TIM\_IC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Init ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li><b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 46.2.17.2 HAL\_TIM\_IC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_DeInit ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Deinitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 46.2.17.3 HAL\_TIM\_IC\_MspInit

Function Name	<b>void HAL_TIM_IC_MspInit ( <i>TIM_HandleTypeDef</i> * htim)</b>
---------------	---

Function Description	Initializes the TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.17.4 HAL\_TIM\_IC\_MspDeInit

Function Name	<b>void HAL_TIM_IC_MspDeInit ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Deinitializes TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

Notes

• None.

#### 46.2.17.5 HAL\_TIM\_IC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start ( <i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b>TIM_CHANNEL_1</b> : TIM Channel 1 selected</li> <li>– <b>TIM_CHANNEL_2</b> : TIM Channel 2 selected</li> <li>– <b>TIM_CHANNEL_3</b> : TIM Channel 3 selected</li> <li>– <b>TIM_CHANNEL_4</b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

Notes

• None.

#### 46.2.17.6 HAL\_TIM\_IC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, uint32_t <i>Channel</i>)</b>
Function Description	Stops the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected</li> <li>- <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected</li> <li>- <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected</li> <li>- <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.17.7 HAL\_TIM\_IC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_IT ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, uint32_t <i>Channel</i>)</b>
Function Description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <i>TIM_CHANNEL_1</i> : TIM Channel 1 selected</li> <li>- <i>TIM_CHANNEL_2</i> : TIM Channel 2 selected</li> <li>- <i>TIM_CHANNEL_3</i> : TIM Channel 3 selected</li> <li>- <i>TIM_CHANNEL_4</i> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.17.8 HAL\_TIM\_IC\_Stop\_IT

---

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel</i></b> )
Function Description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.17.9 HAL\_TIM\_IC\_Start\_DMA

---

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length</i></b> )
Function Description	Starts the TIM Input Capture measurement on in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> <li>• <b>pData</b> : The destination Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from TIM peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.17.10 HAL\_TIM\_IC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Input Capture measurement on in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>– <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>– <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 46.2.18 Time One Pulse functions

### 46.2.18.1 HAL\_TIM\_OnePulse\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Init (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t OnePulseMode)</b>
Function Description	Initializes the TIM One Pulse Time Base according to the specified parameters in the <i>TIM_HandleTypeDef</i> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>OnePulseMode</b> : Select the One pulse mode. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_OPMODE_SINGLE</i></b> : Only one pulse will be generated.</li> <li>– <b><i>TIM_OPMODE_REPETITIVE</i></b> : Repetitive pulses wil be generated.</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 46.2.18.2 HAL\_TIM\_OnePulse\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_DelInit (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	DeInitializes the TIM One Pulse.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.18.3 HAL\_TIM\_OnePulse\_MspInit

Function Name	<b>void HAL_TIM_OnePulse_MspInit (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	Initializes the TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.18.4 HAL\_TIM\_OnePulse\_MspDelInit

Function Name	<b>void HAL_TIM_OnePulse_MspDelInit (</b> <i>TIM_HandleTypeDef * htim)</i>
Function Description	DeInitializes TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.18.5 HAL\_TIM\_OnePulse\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t OutputChannel)</b>
Function Description	Starts the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>OutputChannel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.18.6 HAL\_TIM\_OnePulse\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t OutputChannel)</b>
Function Description	Stops the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>OutputChannel</b> : TIM Channels to be disable. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.18.7 HAL\_TIM\_OnePulse\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t OutputChannel)</b>
Function Description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>OutputChannel</b> : TIM Channels to be enabled. This</li> </ul>

	parameter can be one of the following values:
	– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected
	– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected
Return values	• <b>HAL status</b>
Notes	• None.

#### 46.2.18.8 HAL\_TIM\_OnePulse\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t OutputChannel)</b>
Function Description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>OutputChannel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 46.2.19 Time Encoder functions

##### 46.2.19.1 HAL\_TIM\_Encoder\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Init (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_Encoder_InitTypeDef</i> * <b>sConfig</b>)</b>
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>sConfig</b> : TIM Encoder Interface configuration structure</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 46.2.19.2 HAL\_TIM\_Encoder\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_DelInit (</b> <i>TIM_HandleTypeDef</i> * <b>htim)</b>
Function Description	Deinitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.19.3 HAL\_TIM\_Encoder\_MspInit

Function Name	<b>void HAL_TIM_Encoder_MspInit (</b> <i>TIM_HandleTypeDef</i> * <b>htim)</b>
Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.19.4 HAL\_TIM\_Encoder\_MspDelInit

Function Name	<b>void HAL_TIM_Encoder_MspDelInit (</b> <i>TIM_HandleTypeDef</i> * <b>htim)</b>
Function Description	Deinitializes TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.19.5 HAL\_TIM\_Encoder\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li><li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values:<ul style="list-style-type: none"><li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li><li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.19.6 HAL\_TIM\_Encoder\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li><li>• <b>Channel</b> : TIM Channels to be disabled. This parameter can be one of the following values:<ul style="list-style-type: none"><li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li><li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li></ul></li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.19.7 HAL\_TIM\_Encoder\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel</i></b> )
Function Description	Starts the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.19.8 HAL\_TIM\_Encoder\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel</i></b> )
Function Description	Stops the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.19.9 HAL\_TIM\_Encoder\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length</i></b> )
Function Description	Starts the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul> </li> <li>• <b>pData1</b> : The destination Buffer address for IC1.</li> <li>• <b>pData2</b> : The destination Buffer address for IC2.</li> <li>• <b>Length</b> : The length of data to be transferred from TIM peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.19.10 HAL\_TIM\_Encoder\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>– <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 46.2.20 TIM IRQ handler management

#### 46.2.20.1 HAL\_TIM\_IRQHandler

Function Name	<b>void HAL_TIM_IRQHandler (</b> <b><i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	This function handles TIM interrupts requests.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 46.2.21 Peripheral Control functions

### 46.2.21.1 HAL\_TIM\_OC\_ConfigChannel

Function Name	<code>HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (</code> <code>    TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig,</code> <code>    uint32_t Channel)</code>
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the <code>TIM_OC_InitTypeDef</code> .
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> <li>• <b>sConfig</b> : TIM Output Compare configuration structure</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>TIM_CHANNEL_1</code> : TIM Channel 1 selected</li> <li>– <code>TIM_CHANNEL_2</code> : TIM Channel 2 selected</li> <li>– <code>TIM_CHANNEL_3</code> : TIM Channel 3 selected</li> <li>– <code>TIM_CHANNEL_4</code> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 46.2.21.2 HAL\_TIM\_IC\_ConfigChannel

Function Name	<code>HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (</code> <code>    TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig,</code> <code>    uint32_t Channel)</code>
Function Description	Initializes the TIM Input Capture Channels according to the specified parameters in the <code>TIM_IC_InitTypeDef</code> .
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> <li>• <b>sConfig</b> : TIM Input Capture configuration structure</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>TIM_CHANNEL_1</code> : TIM Channel 1 selected</li> <li>– <code>TIM_CHANNEL_2</code> : TIM Channel 2 selected</li> <li>– <code>TIM_CHANNEL_3</code> : TIM Channel 3 selected</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.3 HAL\_TIM\_PWM\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_OC_InitTypeDef</i> * sConfig,</b> <b><i>uint32_t</i> Channel)</b>
Function Description	Initializes the TIM PWM channels according to the specified parameters in the <i>TIM_OC_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>sConfig</b> : TIM PWM configuration structure</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.4 HAL\_TIM\_OnePulse\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_OnePulse_InitTypeDef</i> *</b> <b><i>sConfig, uint32_t OutputChannel, uint32_t InputChannel</i></b> )
Function Description	Initializes the TIM One Pulse Channels according to the specified parameters in the <i>TIM_OnePulse_InitTypeDef</i> .
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>sConfig</b> : TIM One Pulse configuration structure</li> <li>• <b>OutputChannel</b> : TIM Channels to be enabled. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>InputChannel</b> : TIM Channels to be enabled. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.5 HAL\_TIM\_DMABurst\_WriteStart

Function Name	<code>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (</code> <b><i>TIM_HandleTypeDef</i></b> * <i>htim</i> , <i>uint32_t</i> <i>BurstBaseAddress</i> , <i>uint32_t</i> <i>BurstRequestSrc</i> , <i>uint32_t</i> * <i>BurstBuffer</i> , <i>uint32_t</i> <b><i>BurstLength</i></b> )
Function Description	Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b><i>TIM_HandleTypeDef</i></b> structure that contains the configuration information for TIM module.</li> <li>• <b>BurstBaseAddress</b> : TIM Base address from when the DMA will starts the Data write. This parameters can be on of the following values:           <ul style="list-style-type: none"> <li>- <b><i>TIM_DMABase_CR1</i></b> :</li> <li>- <b><i>TIM_DMABase_CR2</i></b> :</li> <li>- <b><i>TIM_DMABase_SMCR</i></b> :</li> <li>- <b><i>TIM_DMABase_DIER</i></b> :</li> <li>- <b><i>TIM_DMABase_SR</i></b> :</li> <li>- <b><i>TIM_DMABase_EGR</i></b> :</li> <li>- <b><i>TIM_DMABase_CCMR1</i></b> :</li> <li>- <b><i>TIM_DMABase_CCMR2</i></b> :</li> <li>- <b><i>TIM_DMABase_CCER</i></b> :</li> <li>- <b><i>TIM_DMABase_CNT</i></b> :</li> <li>- <b><i>TIM_DMABase_PSC</i></b> :</li> <li>- <b><i>TIM_DMABase_ARR</i></b> :</li> <li>- <b><i>TIM_DMABase_RCR</i></b> :</li> <li>- <b><i>TIM_DMABase_CCR1</i></b> :</li> <li>- <b><i>TIM_DMABase_CCR2</i></b> :</li> <li>- <b><i>TIM_DMABase_CCR3</i></b> :</li> <li>- <b><i>TIM_DMABase_CCR4</i></b> :</li> <li>- <b><i>TIM_DMABase_BDTR</i></b> :</li> <li>- <b><i>TIM_DMABase_DCR</i></b> :</li> </ul> </li> <li>• <b>BurstRequestSrc</b> : TIM DMA Request sources. This parameters can be on of the following values:           <ul style="list-style-type: none"> <li>- <b><i>TIM_DMA_UPDATE</i></b> : TIM update Interrupt source</li> <li>- <b><i>TIM_DMA_CC1</i></b> : TIM Capture Compare 1 DMA source</li> <li>- <b><i>TIM_DMA_CC2</i></b> : TIM Capture Compare 2 DMA source</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>- <b><i>TIM_DMA_CC3</i></b> : TIM Capture Compare 3 DMA source</li> <li>- <b><i>TIM_DMA_CC4</i></b> : TIM Capture Compare 4 DMA source</li> <li>- <b><i>TIM_DMA_COM</i></b> : TIM Commutation DMA source</li> <li>- <b><i>TIM_DMA_TRIGGER</i></b> : TIM Trigger DMA source</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.6 HAL\_TIM\_DMABurst\_WriteStop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t BurstRequestSrc)</b>
Function Description	Stops the TIM DMA Burst mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>BurstRequestSrc</b> : TIM DMA Request sources to disable</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.7 HAL\_TIM\_DMABurst\_ReadStart

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t BurstBaseAddress,</b> <b>uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t</b> <b>BurstLength)</b>
Function Description	Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>BurstBaseAddress</b> : TIM Base address from when the DMA will starts the Data read. This parameters can be one of the following values: <ul style="list-style-type: none"> <li>- <b><i>TIM_DMABase_CR1</i></b> :</li> </ul> </li> </ul>

- ***TIM\_DMABase\_CR2*** :
  - ***TIM\_DMABase\_SMCR*** :
  - ***TIM\_DMABase\_DIER*** :
  - ***TIM\_DMABase\_SR*** :
  - ***TIM\_DMABase\_EGR*** :
  - ***TIM\_DMABase\_CCMR1*** :
  - ***TIM\_DMABase\_CCMR2*** :
  - ***TIM\_DMABase\_CCER*** :
  - ***TIM\_DMABase\_CNT*** :
  - ***TIM\_DMABase\_PSC*** :
  - ***TIM\_DMABase\_ARR*** :
  - ***TIM\_DMABase\_RCR*** :
  - ***TIM\_DMABase\_CCR1*** :
  - ***TIM\_DMABase\_CCR2*** :
  - ***TIM\_DMABase\_CCR3*** :
  - ***TIM\_DMABase\_CCR4*** :
  - ***TIM\_DMABase\_BDTR*** :
  - ***TIM\_DMABase\_DCR*** :
  - **BurstRequestSrc** : TIM DMA Request sources. This parameters can be 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 | • <b>HAL status</b> |
| Notes         | • None.             |

#### 46.2.21.8 HAL\_TIM\_DMABurst\_ReadStop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t BurstRequestSrc)</b>
Function Description	Stop the DMA burst reading.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>BurstRequestSrc</b> : TIM DMA Request sources to disable.</li> </ul>
Return values	• <b>HAL status</b>

- |       |   |
|-------|---|
| Notes | <ul style="list-style-type: none"> <li>• None.</li> </ul> |
|-------|---|

#### 46.2.21.9 HAL\_TIM\_GenerateEvent

Function Name	<b>HAL_StatusTypeDef HAL_TIM_GenerateEvent (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t EventSource)</b>
Function Description	Generate a software event.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>EventSource</b> : specifies the event source. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>TIM_EventSource_Update</i></b> : Timer update Event source</li> <li>– <b><i>TIM_EventSource_CC1</i></b> : Timer Capture Compare 1 Event source</li> <li>– <b><i>TIM_EventSource_CC2</i></b> : Timer Capture Compare 2 Event source</li> <li>– <b><i>TIM_EventSource_CC3</i></b> : Timer Capture Compare 3 Event source</li> <li>– <b><i>TIM_EventSource_CC4</i></b> : Timer Capture Compare 4 Event source</li> <li>– <b><i>TIM_EventSource_COM</i></b> : Timer COM event source</li> <li>– <b><i>TIM_EventSource_Trigger</i></b> : Timer Trigger Event source</li> <li>– <b><i>TIM_EventSource_Break</i></b> : Timer Break event source</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• TIM6 and TIM7 can only generate an update event.</li> <li>• <i>TIM_EventSource_COM</i> and <i>TIM_EventSource_Break</i> are used only with TIM1 and TIM8.</li> </ul>

#### 46.2.21.10 HAL\_TIM\_ConfigOCrefClear

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (</b> <b><i>TIM_HandleTypeDef</i> * htim, <i>TIM_ClearInputConfigTypeDef</i> * sClearInputConfig, uint32_t Channel)</b>
Function Description	Configures the OCRef clear feature.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that</li> </ul>

---

	<p>contains the configuration information for TIM module.</p> <ul style="list-style-type: none"> <li>• <b>sClearInputConfig</b> : pointer to a <code>TIM_ClearInputConfigTypeDef</code> structure that contains the OCREF clear feature and parameters for the TIM peripheral.</li> <li>• <b>Channel</b> : specifies the TIM Channel. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>TIM_CHANNEL_1</code> : TIM Channel 1 selected</li> <li>– <code>TIM_CHANNEL_2</code> : TIM Channel 2 selected</li> <li>– <code>TIM_CHANNEL_3</code> : TIM Channel 3 selected</li> <li>– <code>TIM_CHANNEL_4</code> : TIM Channel 4 selected</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.11 HAL\_TIM\_ConfigClockSource

Function Name	<code>HAL_StatusTypeDef HAL_TIM_ConfigClockSource (</code> <code>    TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef *</code> <code>    sClockSourceConfig)</code>
Function Description	Configures the clock source to be used.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> <li>• <b>sClockSourceConfig</b> : pointer to a <code>TIM_ClockConfigTypeDef</code> structure that contains the clock source information for the TIM peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.12 HAL\_TIM\_ConfigTI1Input

Function Name	<code>HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (</code> <code>    TIM_HandleTypeDef * htim, uint32_t TI1_Selection)</code>
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	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module..</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>TI1_Selection :</b> Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b><i>TIM_TI1SELECTION_CH1</i></b> : The TIMx_CH1 pin is connected to TI1 input</li> <li>– <b><i>TIM_TI1SELECTION_XORCOMBINATION</i></b> : The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.13 HAL\_TIM\_SlaveConfigSynchronization

Function Name	<b>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, <i>TIM_SlaveConfigTypeDef</i> * <i>sSlaveConfig</i>)</b>
Function Description	Configures the TIM in Slave mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module..</li> <li>• <b>sSlaveConfig</b> : pointer to a <i>TIM_SlaveConfigTypeDef</i> 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).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 46.2.21.14 HAL\_TIM\_ReadCapturedValue

Function Name	<b>uint32_t HAL_TIM_ReadCapturedValue ( <i>TIM_HandleTypeDef</i> * <i>htim</i>, uint32_t <i>Channel</i>)</b>
Function Description	Read the captured value from Capture Compare unit.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module..</li> <li>• <b>Channel</b> : TIM Channels to be enabled. This parameter can be one of the following values:</li> </ul>

---

	<ul style="list-style-type: none"> <li>- <b><i>TIM_CHANNEL_1</i></b> : TIM Channel 1 selected</li> <li>- <b><i>TIM_CHANNEL_2</i></b> : TIM Channel 2 selected</li> <li>- <b><i>TIM_CHANNEL_3</i></b> : TIM Channel 3 selected</li> <li>- <b><i>TIM_CHANNEL_4</i></b> : TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>Captured value</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 46.2.22 TIM Callbacks functions

### 46.2.22.1 HAL\_TIM\_PeriodElapsedCallback

Function Name	<b>void HAL_TIM_PeriodElapsedCallback ( <i>TIM_HandleTypeDef</i> * <b>htim</b>)</b>
Function Description	Period elapsed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 46.2.22.2 HAL\_TIM\_OC\_DelayElapsedCallback

Function Name	<b>void HAL_TIM_OC_DelayElapsedCallback ( <i>TIM_HandleTypeDef</i> * <b>htim</b>)</b>
Function Description	Output Compare callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 46.2.22.3 HAL\_TIM\_IC\_CaptureCallback

Function Name	<b>void HAL_TIM_IC_CaptureCallback ( <i>TIM_HandleTypeDef</i> * <i>htim</i>)</b>
Function Description	Input Capture callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 46.2.22.4 HAL\_TIM\_PWM\_PulseFinishedCallback

Function Name	<b>void HAL_TIM_PWM_PulseFinishedCallback ( <i>TIM_HandleTypeDef</i> * <i>htim</i>)</b>
Function Description	PWM Pulse finished callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 46.2.22.5 HAL\_TIM\_TriggerCallback

Function Name	<b>void HAL_TIM_TriggerCallback ( <i>TIM_HandleTypeDef</i> * <i>htim</i>)</b>
Function Description	Hall Trigger detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 46.2.22.6 HAL\_TIM\_ErrorCallback

Function Name	<b>void HAL_TIM_ErrorCallback ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Timer error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 46.3 TIM Firmware driver defines

### 46.3.1 TIM

TIM

*TIM\_Channel*

- #define: ***TIM\_CHANNEL\_1* ((uint32\_t)0x0000)**
- #define: ***TIM\_CHANNEL\_2* ((uint32\_t)0x0004)**
- #define: ***TIM\_CHANNEL\_3* ((uint32\_t)0x0008)**
- #define: ***TIM\_CHANNEL\_4* ((uint32\_t)0x000C)**
- #define: ***TIM\_CHANNEL\_ALL* ((uint32\_t)0x0018)**

*TIM\_ClearInput\_Polarity*

- #define: ***TIM\_CLEARINPUTPOLARITY\_INVERTED***  
***TIM\_ETRPOLARITY\_INVERTED***

Polarity for ETRx pin



- #define: **TIM\_CLEARINPUTPOLARITY\_NONINVERTED**  
**TIM\_ETRPOLARITY\_NONINVERTED**

Polarity for ETRx pin

#### ***TIM\_ClearInput\_Prescaler***

- #define: **TIM\_CLEARINPUTPRESCALER\_DIV1** **TIM\_ETRPRESCALER\_DIV1**

No prescaler is used

- #define: **TIM\_CLEARINPUTPRESCALER\_DIV2** **TIM\_ETRPRESCALER\_DIV2**

Prescaler for External ETR pin: Capture performed once every 2 events.

- #define: **TIM\_CLEARINPUTPRESCALER\_DIV4** **TIM\_ETRPRESCALER\_DIV4**

Prescaler for External ETR pin: Capture performed once every 4 events.

- #define: **TIM\_CLEARINPUTPRESCALER\_DIV8** **TIM\_ETRPRESCALER\_DIV8**

Prescaler for External ETR pin: Capture performed once every 8 events.

#### ***TIM\_ClearInput\_Source***

- #define: **TIM\_CLEARINPUTSOURCE\_ETR** ((*uint32\_t*)0x0001)

- #define: **TIM\_CLEARINPUTSOURCE\_NONE** ((*uint32\_t*)0x0000)

#### ***TIM\_ClockDivision***

- #define: **TIM\_CLOCKDIVISION\_DIV1** ((*uint32\_t*)0x0000)

- #define: **TIM\_CLOCKDIVISION\_DIV2** (**TIM\_CR1\_CKD\_0**)

- #define: **TIM\_CLOCKDIVISION\_DIV4** (**TIM\_CR1\_CKD\_1**)

#### ***TIM\_Clock\_Polarity***

- #define: **TIM\_CLOCKPOLARITY\_INVERTED** **TIM\_ETRPOLARITY\_INVERTED**

*Polarity for ETRx clock sources*

- #define: **TIM\_CLOCKPOLARITY\_NONINVERTED**  
**TIM\_ETRPOLARITY\_NONINVERTED**

*Polarity for ETRx clock sources*

- #define: **TIM\_CLOCKPOLARITY\_RISING**  
**TIM\_INPUTCHANNELPOLARITY\_RISING**

*Polarity for Tlx clock sources*

- #define: **TIM\_CLOCKPOLARITY\_FALLING**  
**TIM\_INPUTCHANNELPOLARITY\_FALLING**

*Polarity for Tlx clock sources*

- #define: **TIM\_CLOCKPOLARITY\_BOTHEDGE**  
**TIM\_INPUTCHANNELPOLARITY\_BOTHEDGE**

*Polarity for Tlx clock sources*

#### ***TIM\_Clock\_Prescaler***

- #define: **TIM\_CLOCKPRESALER\_DIV1** **TIM\_ETRPRESALER\_DIV1**

*No prescaler is used*

- #define: **TIM\_CLOCKPRESALER\_DIV2** **TIM\_ETRPRESALER\_DIV2**

*Prescaler for External ETR Clock: Capture performed once every 2 events.*

- #define: **TIM\_CLOCKPRESALER\_DIV4** **TIM\_ETRPRESALER\_DIV4**

*Prescaler for External ETR Clock: Capture performed once every 4 events.*

- #define: **TIM\_CLOCKPRESALER\_DIV8** **TIM\_ETRPRESALER\_DIV8**

*Prescaler for External ETR Clock: Capture performed once every 8 events.*

#### ***TIM\_Clock\_Source***

- #define: **TIM\_CLOCKSOURCE\_ETRMODE2** (**TIM\_SMCR\_ETPS\_1**)

- #define: **TIM\_CLOCKSOURCE\_INTERNAL** (**TIM\_SMCR\_ETPS\_0**)

- #define: ***TIM\_CLOCKSOURCE\_ITR0 ((uint32\_t)0x0000)***
- #define: ***TIM\_CLOCKSOURCE\_ITR1 (TIM\_SMCR\_TS\_0)***
- #define: ***TIM\_CLOCKSOURCE\_ITR2 (TIM\_SMCR\_TS\_1)***
- #define: ***TIM\_CLOCKSOURCE\_ITR3 (TIM\_SMCR\_TS\_0 | TIM\_SMCR\_TS\_1)***
- #define: ***TIM\_CLOCKSOURCE\_TI1ED (TIM\_SMCR\_TS\_2)***
- #define: ***TIM\_CLOCKSOURCE\_TI1 (TIM\_SMCR\_TS\_0 | TIM\_SMCR\_TS\_2)***
- #define: ***TIM\_CLOCKSOURCE\_TI2 (TIM\_SMCR\_TS\_1 | TIM\_SMCR\_TS\_2)***
- #define: ***TIM\_CLOCKSOURCE\_ETRMODE1 (TIM\_SMCR\_TS)***

#### ***TIM\_Counter\_Mode***

- #define: ***TIM\_COUNTERMODE\_UP ((uint32\_t)0x0000)***
- #define: ***TIM\_COUNTERMODE\_DOWN TIM\_CR1\_DIR***
- #define: ***TIM\_COUNTERMODE\_CENTERALIGNED1 TIM\_CR1\_CMS\_0***
- #define: ***TIM\_COUNTERMODE\_CENTERALIGNED2 TIM\_CR1\_CMS\_1***

- #define: ***TIM\_COUNTERMODE\_CENTERALIGNED3 TIM\_CR1\_CMS***

***TIM\_DMA\_Base\_address***

- #define: ***TIM\_DMABase\_CR1 (0x00000000)***
- #define: ***TIM\_DMABase\_CR2 (0x00000001)***
- #define: ***TIM\_DMABase\_SMCR (0x00000002)***
- #define: ***TIM\_DMABase\_DIER (0x00000003)***
- #define: ***TIM\_DMABase\_SR (0x00000004)***
- #define: ***TIM\_DMABase\_EGR (0x00000005)***
- #define: ***TIM\_DMABase\_CCMR1 (0x00000006)***
- #define: ***TIM\_DMABase\_CCMR2 (0x00000007)***
- #define: ***TIM\_DMABase\_CCER (0x00000008)***
- #define: ***TIM\_DMABase\_CNT (0x00000009)***
- #define: ***TIM\_DMABase\_PSC (0x0000000A)***

- #define: ***TIM\_DMABase\_ARR (0x0000000B)***
  - #define: ***TIM\_DMABase\_RCR (0x0000000C)***
  - #define: ***TIM\_DMABase\_CCR1 (0x0000000D)***
  - #define: ***TIM\_DMABase\_CCR2 (0x0000000E)***
  - #define: ***TIM\_DMABase\_CCR3 (0x0000000F)***
  - #define: ***TIM\_DMABase\_CCR4 (0x00000010)***
  - #define: ***TIM\_DMABase\_BDTR (0x00000011)***
  - #define: ***TIM\_DMABase\_DCR (0x00000012)***
  - #define: ***TIM\_DMABase\_OR (0x00000013)***
- TIM\_DMA\_Burst\_Length***
- #define: ***TIM\_DMABurstLength\_1Transfer (0x00000000)***
  - #define: ***TIM\_DMABurstLength\_2Transfers (0x00000100)***

- #define: ***TIM\_DMABurstLength\_3Transfers (0x00000200)***
- #define: ***TIM\_DMABurstLength\_4Transfers (0x00000300)***
- #define: ***TIM\_DMABurstLength\_5Transfers (0x00000400)***
- #define: ***TIM\_DMABurstLength\_6Transfers (0x00000500)***
- #define: ***TIM\_DMABurstLength\_7Transfers (0x00000600)***
- #define: ***TIM\_DMABurstLength\_8Transfers (0x00000700)***
- #define: ***TIM\_DMABurstLength\_9Transfers (0x00000800)***
- #define: ***TIM\_DMABurstLength\_10Transfers (0x00000900)***
- #define: ***TIM\_DMABurstLength\_11Transfers (0x00000A00)***
- #define: ***TIM\_DMABurstLength\_12Transfers (0x00000B00)***
- #define: ***TIM\_DMABurstLength\_13Transfers (0x00000C00)***
- #define: ***TIM\_DMABurstLength\_14Transfers (0x00000D00)***

- #define: ***TIM\_DMABurstLength\_15Transfers (0x00000E00)***
- #define: ***TIM\_DMABurstLength\_16Transfers (0x00000F00)***
- #define: ***TIM\_DMABurstLength\_17Transfers (0x00001000)***
- #define: ***TIM\_DMABurstLength\_18Transfers (0x00001100)***

#### ***TIM\_DMA\_sources***

- #define: ***TIM\_DMA\_UPDATE (TIM\_DIER\_UDE)***
- #define: ***TIM\_DMA\_CC1 (TIM\_DIER\_CC1DE)***
- #define: ***TIM\_DMA\_CC2 (TIM\_DIER\_CC2DE)***
- #define: ***TIM\_DMA\_CC3 (TIM\_DIER\_CC3DE)***
- #define: ***TIM\_DMA\_CC4 (TIM\_DIER\_CC4DE)***
- #define: ***TIM\_DMA\_COM (TIM\_DIER\_COMDE)***
- #define: ***TIM\_DMA\_TRIGGER (TIM\_DIER\_TDE)***

#### ***TIM\_Encoder\_Mode***

- #define: ***TIM\_ENCODERMODE\_TI1 (TIM\_SMCR\_SMS\_0)***

- #define: **TIM\_ENCODERMODE\_TI2 (TIM\_SMCR\_SMS\_1)**
- #define: **TIM\_ENCODERMODE\_TI12 (TIM\_SMCR\_SMS\_1 | TIM\_SMCR\_SMS\_0)**

#### ***TIM\_ETR\_Polarity***

- #define: **TIM\_ETRPOLARITY\_INVERTED (TIM\_SMCR\_ETP)**  
*Polarity for ETR source*
- #define: **TIM\_ETRPOLARITY\_NONINVERTED ((uint32\_t)0x0000)**  
*Polarity for ETR source*

#### ***TIM\_ETR\_Prescaler***

- #define: **TIM\_ETRPRESCALER\_DIV1 ((uint32\_t)0x0000)**  
*No prescaler is used*
- #define: **TIM\_ETRPRESCALER\_DIV2 (TIM\_SMCR\_ETPS\_0)**  
*ETR input source is divided by 2*
- #define: **TIM\_ETRPRESCALER\_DIV4 (TIM\_SMCR\_ETPS\_1)**  
*ETR input source is divided by 4*
- #define: **TIM\_ETRPRESCALER\_DIV8 (TIM\_SMCR\_ETPS)**  
*ETR input source is divided by 8*

#### ***TIM\_Event\_Source***

- #define: **TIM\_EventSource\_Update TIM\_EGR\_UG**
- #define: **TIM\_EventSource\_CC1 TIM\_EGR\_CC1G**
- #define: **TIM\_EventSource\_CC2 TIM\_EGR\_CC2G**

- #define: ***TIM\_EventSource\_CC3 TIM\_EGR\_CC3G***
- #define: ***TIM\_EventSource\_CC4 TIM\_EGR\_CC4G***
- #define: ***TIM\_EventSource\_COM TIM\_EGR\_COMG***
- #define: ***TIM\_EventSource\_Trigger TIM\_EGR\_TG***
- #define: ***TIM\_EventSource\_Break TIM\_EGR\_BG***

#### ***TIM\_Flag\_definition***

- #define: ***TIM\_FLAG\_UPDATE (TIM\_SR UIF)***
- #define: ***TIM\_FLAG\_CC1 (TIM\_SR\_CC1IF)***
- #define: ***TIM\_FLAG\_CC2 (TIM\_SR\_CC2IF)***
- #define: ***TIM\_FLAG\_CC3 (TIM\_SR\_CC3IF)***
- #define: ***TIM\_FLAG\_CC4 (TIM\_SR\_CC4IF)***
- #define: ***TIM\_FLAG\_COM (TIM\_SR\_COMIF)***

- #define: ***TIM\_FLAG\_TRIGGER (TIM\_SR\_TIF)***
- #define: ***TIM\_FLAG\_BREAK (TIM\_SR\_BIF)***
- #define: ***TIM\_FLAG\_CC1OF (TIM\_SR\_CC1OF)***
- #define: ***TIM\_FLAG\_CC2OF (TIM\_SR\_CC2OF)***
- #define: ***TIM\_FLAG\_CC3OF (TIM\_SR\_CC3OF)***
- #define: ***TIM\_FLAG\_CC4OF (TIM\_SR\_CC4OF)***

#### ***TIM\_Input\_Capture\_Polarity***

- #define: ***TIM\_ICPOLARITY\_RISING TIM\_INPUTCHANNELPOLARITY\_RISING***
- #define: ***TIM\_ICPOLARITY\_FALLING TIM\_INPUTCHANNELPOLARITY\_FALLING***
- #define: ***TIM\_ICPOLARITY\_BOTHEDGE***  
***TIM\_INPUTCHANNELPOLARITY\_BOTHEDGE***

#### ***TIM\_Input\_Capture\_Prescaler***

- #define: ***TIM\_ICPSC\_DIV1 ((uint32\_t)0x0000)***  
*Capture performed each time an edge is detected on the capture input*
- #define: ***TIM\_ICPSC\_DIV2 (TIM\_CCMR1\_IC1PSC\_0)***  
*Capture performed once every 2 events*
- #define: ***TIM\_ICPSC\_DIV4 (TIM\_CCMR1\_IC1PSC\_1)***

*Capture performed once every 4 events*

- #define: **TIM\_ICPSC\_DIV8 (TIM\_CCMR1\_IC1PSC)**

*Capture performed once every 8 events*

#### ***TIM\_Input\_Capture\_Selection***

- #define: **TIM\_ICSELECTION\_DIRECTTI (TIM\_CCMR1\_CC1S\_0)**

*TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively*

- #define: **TIM\_ICSELECTION\_INDIRECTTI (TIM\_CCMR1\_CC1S\_1)**

*TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively*

- #define: **TIM\_ICSELECTION\_TRC (TIM\_CCMR1\_CC1S)**

*TIM Input 1, 2, 3 or 4 is selected to be connected to TRC*

#### ***TIM\_Input\_Channel\_Polarity***

- #define: **TIM\_INPUTCHANNELPOLARITY\_RISING ((uint32\_t)0x00000000)**

*Polarity for TIx source*

- #define: **TIM\_INPUTCHANNELPOLARITY\_FALLING (TIM\_CCER\_CC1P)**

*Polarity for TIx source*

- #define: **TIM\_INPUTCHANNELPOLARITY\_BOTHEDGE (TIM\_CCER\_CC1P | TIM\_CCER\_CC1NP)**

*Polarity for TIx source*

#### ***TIM Interrupt\_definition***

- #define: **TIM\_IT\_UPDATE (TIM\_DIER\_UIE)**

- #define: **TIM\_IT\_CC1 (TIM\_DIER\_CC1IE)**

- #define: **TIM\_IT\_CC2 (TIM\_DIER\_CC2IE)**

- #define: **TIM\_IT\_CC3 (TIM\_DIER\_CC3IE)**

- #define: ***TIM\_IT\_CC4 (TIM\_DIER\_CC4IE)***
- #define: ***TIM\_IT\_COM (TIM\_DIER\_COMIE)***
- #define: ***TIM\_IT\_TRIGGER (TIM\_DIER\_TIE)***
- #define: ***TIM\_IT\_BREAK (TIM\_DIER\_BIE)***

#### ***TIM\_One\_Pulse\_Mode***

- #define: ***TIM\_OPMODE\_SINGLE (TIM\_CR1\_OPM)***
- #define: ***TIM\_OPMODE\_REPEATITIVE ((uint32\_t)0x0000)***

#### ***TIM\_Output\_Compare\_and\_PWM\_modes***

- #define: ***TIM\_OCMODE\_TIMING ((uint32\_t)0x0000)***
- #define: ***TIM\_OCMODE\_ACTIVE (TIM\_CCMR1\_OC1M\_0)***
- #define: ***TIM\_OCMODE\_INACTIVE (TIM\_CCMR1\_OC1M\_1)***
- #define: ***TIM\_OCMODE\_TOGGLE (TIM\_CCMR1\_OC1M\_0 | TIM\_CCMR1\_OC1M\_1)***
- #define: ***TIM\_OCMODE\_PWM1 (TIM\_CCMR1\_OC1M\_1 | TIM\_CCMR1\_OC1M\_2)***

- #define: ***TIM\_OCMODE\_PWM2 (TIM\_CCMR1\_OC1M)***
- #define: ***TIM\_OCMODE\_FORCED\_ACTIVE (TIM\_CCMR1\_OC1M\_0 | TIM\_CCMR1\_OC1M\_2)***
- #define: ***TIM\_OCMODE\_FORCED\_INACTIVE (TIM\_CCMR1\_OC1M\_2)***

#### ***TIM\_Output\_Compare\_Idle\_State***

- #define: ***TIM\_OCIDLESTATE\_SET (TIM\_CR2\_OIS1)***
- #define: ***TIM\_OCIDLESTATE\_RESET ((uint32\_t)0x0000)***

#### ***TIM\_Output\_Compare\_N\_Idle\_State***

- #define: ***TIM\_OCNIDLESTATE\_SET (TIM\_CR2\_OIS1N)***
- #define: ***TIM\_OCNIDLESTATE\_RESET ((uint32\_t)0x0000)***

#### ***TIM\_Output\_Compare\_N\_Polarity***

- #define: ***TIM\_OCNPOLARITY\_HIGH ((uint32\_t)0x0000)***
- #define: ***TIM\_OCNPOLARITY\_LOW (TIM\_CCER\_CC1NP)***

#### ***TIM\_Output\_Compare\_N\_State***

- #define: ***TIM\_OUTPUTNSTATE\_DISABLE ((uint32\_t)0x0000)***
- #define: ***TIM\_OUTPUTNSTATE\_ENABLE (TIM\_CCER\_CC1NE)***

***TIM\_Output\_Compare\_Polarity***

- #define: ***TIM\_OCPOLARITY\_HIGH ((uint32\_t)0x0000)***

- #define: ***TIM\_OCPOLARITY\_LOW (TIM\_CCER\_CC1P)***

***TIM\_Output\_Compare\_State***

- #define: ***TIM\_OUTPUTSTATE\_DISABLE ((uint32\_t)0x0000)***

- #define: ***TIM\_OUTPUTSTATE\_ENABLE (TIM\_CCER\_CC1E)***

***TIM\_Output\_Fast\_State***

- #define: ***TIM\_OCFAST\_DISABLE ((uint32\_t)0x0000)***

- #define: ***TIM\_OCFAST\_ENABLE (TIM\_CCMR1\_OC1FE)***

***TIM\_Slave\_Mode***

- #define: ***TIM\_SLAVEMODE\_DISABLE ((uint32\_t)0x0000)***

- #define: ***TIM\_SLAVEMODE\_RESET ((uint32\_t)0x0004)***

- #define: ***TIM\_SLAVEMODE\_GATED ((uint32\_t)0x0005)***

- #define: ***TIM\_SLAVEMODE\_TRIGGER ((uint32\_t)0x0006)***

- #define: **TIM\_SLAVEMODE\_EXTERNAL1** ((*uint32\_t*)0x0007)

#### ***TIM\_TI1\_Selection***

- #define: **TIM\_TI1SELECTION\_CH1** ((*uint32\_t*)0x0000)
- #define: **TIM\_TI1SELECTION\_XORCOMBINATION** (**TIM\_CR2\_TI1S**)

#### ***TIM\_Trigger\_Polarity***

- #define: **TIM\_TRIGGERPOLARITY\_INVERTED** **TIM\_ETRPOLARITY\_INVERTED**  
*Polarity for ETRx trigger sources*
- #define: **TIM\_TRIGGERPOLARITY\_NONINVERTED**  
**TIM\_ETRPOLARITY\_NONINVERTED**  
*Polarity for ETRx trigger sources*
- #define: **TIM\_TRIGGERPOLARITY\_RISING**  
**TIM\_INPUTCHANNELPOLARITY\_RISING**  
*Polarity for TIxFPx or TI1\_ED trigger sources*
- #define: **TIM\_TRIGGERPOLARITY\_FALLING**  
**TIM\_INPUTCHANNELPOLARITY\_FALLING**  
*Polarity for TIxFPx or TI1\_ED trigger sources*
- #define: **TIM\_TRIGGERPOLARITY\_BOTHEDGE**  
**TIM\_INPUTCHANNELPOLARITY\_BOTHEDGE**  
*Polarity for TIxFPx or TI1\_ED trigger sources*

#### ***TIM\_Trigger\_Prescaler***

- #define: **TIM\_TRIGGERPRESCALER\_DIV1** **TIM\_ETRPRESCALER\_DIV1**  
*No prescaler is used*
- #define: **TIM\_TRIGGERPRESCALER\_DIV2** **TIM\_ETRPRESCALER\_DIV2**  
*Prescaler for External ETR Trigger: Capture performed once every 2 events.*
- #define: **TIM\_TRIGGERPRESCALER\_DIV4** **TIM\_ETRPRESCALER\_DIV4**

*Prescaler for External ETR Trigger: Capture performed once every 4 events.*

- #define: ***TIM\_TRIGGERPRESCALER\_DIV8 TIM\_ETRPRESCALER\_DIV8***

*Prescaler for External ETR Trigger: Capture performed once every 8 events.*

#### ***TIM\_Trigger\_Selection***

- #define: ***TIM\_TS\_ITR0 ((uint32\_t)0x0000)***
- #define: ***TIM\_TS\_ITR1 ((uint32\_t)0x0010)***
- #define: ***TIM\_TS\_ITR2 ((uint32\_t)0x0020)***
- #define: ***TIM\_TS\_ITR3 ((uint32\_t)0x0030)***
- #define: ***TIM\_TS\_TI1F\_ED ((uint32\_t)0x0040)***
- #define: ***TIM\_TS\_TI1FP1 ((uint32\_t)0x0050)***
- #define: ***TIM\_TS\_TI2FP2 ((uint32\_t)0x0060)***
- #define: ***TIM\_TS\_ETRF ((uint32\_t)0x0070)***
- #define: ***TIM\_TS\_NONE ((uint32\_t)0xFFFF)***

## 47 HAL TIM Extension Driver

### 47.1 TIMEx Firmware driver registers structures

#### 47.1.1 TIM\_MasterConfigTypeDef

*TIM\_MasterConfigTypeDef* is defined in the `stm32f4xx_hal_tim_ex.h`

##### Data Fields

- *uint32\_t MasterOutputTrigger*
- *uint32\_t MasterSlaveMode*

##### Field Documentation

- *uint32\_t TIM\_MasterConfigTypeDef::MasterOutputTrigger*
  - Trigger output (TRGO) selection. This parameter can be a value of [\*TIMEx\\_Master\\_Mode\\_Selection\*](#)
- *uint32\_t TIM\_MasterConfigTypeDef::MasterSlaveMode*
  - Master/slave mode selection. This parameter can be a value of [\*TIMEx\\_Master\\_Slave\\_Mode\*](#)

#### 47.1.2 TIM\_HallSensor\_InitTypeDef

*TIM\_HallSensor\_InitTypeDef* is defined in the `stm32f4xx_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

### 47.1.3 TIM\_BreakDeadTimeConfigTypeDef

*TIM\_BreakDeadTimeConfigTypeDef* is defined in the `stm32f4xx_hal_tim_ex.h`

#### Data Fields

- `uint32_t OffStateRunMode`
- `uint32_t OffStateIDLEMode`
- `uint32_t LockLevel`
- `uint32_t DeadTime`
- `uint32_t BreakState`
- `uint32_t BreakPolarity`
- `uint32_t AutomaticOutput`

#### Field Documentation

- `uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode`
  - TIM off state in run mode. This parameter can be a value of `TIMEx_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 `TIMEx_OSSI_Off_State_Selection_for_Idle_mode_state`
- `uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel`
  - TIM Lock level. This parameter can be a value of `TIMEx_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 `TIMEx_Break_Input_enable_disable`
- `uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity`
  - TIM Break input polarity. This parameter can be a value of `TIMEx_Break_Polarity`
- `uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput`
  - TIM Automatic Output Enable state. This parameter can be a value of `TIMEx_AOE_Bit_Set_Reset`

## 47.2 TIMEEx Firmware driver API description

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

### 47.2.1 TIMER Extended features

The Timer Extension features include:



1. Complementary outputs with programmable dead-time for :
  - Input Capture
  - 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

#### 47.2.2 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\_OC\_Start\_IT()
  - Complementary PWM generation : HAL\_TIMEx\_PWMN\_Start(),  
  HAL\_TIMEx\_PWMN\_Start\_DMA(), HAL\_TIMEx\_PWMN\_Start\_IT()
  - Complementary One-pulse mode output : HAL\_TIMEx\_OnePulseN\_Start(),  
  HAL\_TIMEx\_OnePulseN\_Start\_IT()
  - Hall Sensor output : HAL\_TIMEx\_HallSensor\_Start(),  
  HAL\_TIMEx\_HallSensor\_Start\_DMA(), HAL\_TIMEx\_HallSensor\_Start\_IT().

#### 47.2.3 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\_DelInit\(\)`](#)
- [`HAL\_TIMEx\_HallSensor\_MspInit\(\)`](#)
- [`HAL\_TIMEx\_HallSensor\_MspDelInit\(\)`](#)
- [`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\(\)`](#)

#### 47.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:

- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.
- [`HAL\_TIMEx\_OCN\_Start\(\)`](#)
- [`HAL\_TIMEx\_OCN\_Stop\(\)`](#)
- [`HAL\_TIMEx\_OCN\_Start\_IT\(\)`](#)
- [`HAL\_TIMEx\_OCN\_Stop\_IT\(\)`](#)
- [`HAL\_TIMEx\_OCN\_Start\_DMA\(\)`](#)
- [`HAL\_TIMEx\_OCN\_Stop\_DMA\(\)`](#)

#### 47.2.5 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()`

#### 47.2.6 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()`

#### 47.2.7 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 commutation event in case of use of the Hall sensor interface.
- Configure the DMA Burst Mode.
- `HAL_TIMEx_ConfigCommuteEvent()`
- `HAL_TIMEx_ConfigCommuteEvent_IT()`
- `HAL_TIMEx_ConfigCommuteEvent_DMA()`
- `HAL_TIMEx_MasterConfigSynchronization()`
- `HAL_TIMEx_ConfigBreakDeadTime()`
- `HAL_TIMEx_RemapConfig()`

#### 47.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback
- `HAL_TIMEx_CommulationCallback()`
- `HAL_TIMEx_BreakCallback()`

#### 47.2.9 Extension Peripheral State functions

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

- ***HAL\_TIMEx\_HallSensor\_GetState()***

## 47.2.10 Timer Hall Sensor functions

### 47.2.10.1 HAL\_TIMEx\_HallSensor\_Init

Function Name	<b><code>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (</code></b>
	<b><code>TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)</code></b>
Function Description	Initializes the TIM Hall Sensor Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> <li>• <b>sConfig</b> : TIM Hall Sensor configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 47.2.10.2 HAL\_TIMEx\_HallSensor\_DelInit

Function Name	<b><code>HAL_StatusTypeDef HAL_TIMEx_HallSensor_DelInit (</code></b>
	<b><code>TIM_HandleTypeDef * htim)</code></b>
Function Description	Delinitializes the TIM Hall Sensor interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 47.2.10.3 HAL\_TIMEx\_HallSensor\_MspInit

Function Name	<b><code>void HAL_TIMEx_HallSensor_MspInit ( TIM_HandleTypeDef * htim)</code></b>
Function Description	Initializes the TIM Hall Sensor MSP.

Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 47.2.10.4 HAL\_TIMEx\_HallSensor\_MspDeInit

Function Name	<b>void HAL_TIMEx_HallSensor_MspDeInit ( <i>TIM_HandleTypeDef</i> * <b>htim</b>)</b>
Function Description	Deinitializes TIM Hall Sensor MSP.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 47.2.10.5 HAL\_TIMEx\_HallSensor\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start ( <i>TIM_HandleTypeDef</i> * <b>htim</b>)</b>
Function Description	Starts the TIM Hall Sensor Interface.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 47.2.10.6 HAL\_TIMEx\_HallSensor\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (</b>
---------------	--

***TIM\_HandleTypeDef \* htim)***

Function Description	Stops the TIM Hall sensor Interface.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**47.2.10.7 HAL\_TIMEx\_HallSensor\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (</b> <b><i>TIM_HandleTypeDef * htim)</i></b>
Function Description	Starts the TIM Hall Sensor Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**47.2.10.8 HAL\_TIMEx\_HallSensor\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (</b> <b><i>TIM_HandleTypeDef * htim)</i></b>
Function Description	Stops the TIM Hall Sensor Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**47.2.10.9 HAL\_TIMEx\_HallSensor\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length</i></b> )
Function Description	Starts the TIM Hall Sensor Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>pData</b> : The destination Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from TIM peripheral to memory.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.10.10 HAL\_TIMEx\_HallSensor\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (</b> <b><i>TIM_HandleTypeDef * htim</i></b> )
Function Description	Stops the TIM Hall Sensor Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 47.2.11 Timer Complementary Output Compare functions

#### 47.2.11.1 HAL\_TIMEx\_OCN\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel</i></b> )
Function Description	Starts the TIM Output Compare signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be enabled. This parameter can be one of the following values: <b>TIM_CHANNEL_1/</b></li> </ul>

---

	TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.11.2 HAL\_TIMEx\_OCN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel</i></b> )
Function Description	Stops the TIM Output Compare signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: <b>TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4</b></li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.11.3 HAL\_TIMEx\_OCN\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel</i></b> )
Function Description	Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be enabled. This parameter can be one of the following values: <b>TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4</b></li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.11.4 HAL\_TIMEx\_OCN\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel</i></b> )
Function Description	Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: <b>TIM_CHANNEL_1/TIM_CHANNEL_2/TIM_CHANNEL_3/TIM_CHANNEL_4</b></li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.11.5 HAL\_TIMEx\_OCN\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length</i></b> )
Function Description	Starts the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be enabled. This parameter can be one of the following values: <b>TIM_CHANNEL_1/TIM_CHANNEL_2/TIM_CHANNEL_3/TIM_CHANNEL_4</b></li> <li>• <b>pData</b> : The source Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from memory to TIM peripheral</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.11.6 HAL\_TIMEx\_OCN\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (</b> <b>TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li><b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li><b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 47.2.12 Timer Complementary PWM functions

### 47.2.12.1 HAL\_TIMEx\_PWMN\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (</b> <b>TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the PWM signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li><b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li><b>Channel</b> : TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4</li> </ul>
Return values	<b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

### 47.2.12.2 HAL\_TIMEx\_PWMN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (</b> <b>TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li><b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li><b>Channel</b> : TIM Channel to be disabled. This parameter can</li> </ul>

be one of the following values: TIM\_CHANNEL\_1/  
TIM\_CHANNEL\_2/ TIM\_CHANNEL\_3/ TIM\_CHANNEL\_4

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul> |
| Notes         | <ul style="list-style-type: none"> <li>• None.</li> </ul>             |

#### 47.2.12.3 HAL\_TIMEx\_PWMN\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Starts the PWM signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.12.4 HAL\_TIMEx\_PWMN\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (</b> <b><i>TIM_HandleTypeDef</i> * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1/ TIM_CHANNEL_2/ TIM_CHANNEL_3/ TIM_CHANNEL_4</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.12.5 HAL\_TIMEx\_PWMN\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (   <b>TIM_HandleTypeDef</b> * htim, uint32_t Channel, uint32_t *   pData, uint16_t Length)</code>
Function Description	Starts the TIM PWM signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be enabled. This parameter can be one of the following values: <b>TIM_CHANNEL_1</b>/ <b>TIM_CHANNEL_2</b>/<b>TIM_CHANNEL_3</b>/<b>TIM_CHANNEL_4</b></li> <li>• <b>pData</b> : The source Buffer address.</li> <li>• <b>Length</b> : The length of data to be transferred from memory to TIM peripheral</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 47.2.12.6 HAL\_TIMEx\_PWMN\_Stop\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (   <b>TIM_HandleTypeDef</b> * htim, uint32_t Channel)</code>
Function Description	Stops the TIM PWM signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <b>TIM_HandleTypeDef</b> structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b> : TIM Channel to be disabled. This parameter can be one of the following values: <b>TIM_CHANNEL_1</b>/ <b>TIM_CHANNEL_2</b>/<b>TIM_CHANNEL_3</b>/<b>TIM_CHANNEL_4</b></li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

### 47.2.13 Timer Complementary One Pulse functions

#### 47.2.13.1 HAL\_TIMEx\_OnePulseN\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (</b> <b>TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
Function Description	Starts the TIM One Pulse signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>OutputChannel</b> : TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1 / IM_CHANNEL_2</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.13.2 HAL\_TIMEx\_OnePulseN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (</b> <b>TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
Function Description	Stops the TIM One Pulse signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>OutputChannel</b> : TIM Channel to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1 / TIM_CHANNEL_2</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.13.3 HAL\_TIMEx\_OnePulseN\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (</b>
---------------	---

***TIM\_HandleTypeDef \* htim, uint32\_t OutputChannel)***

Function Description	Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> <li>• <b>OutputChannel</b> : TIM Channel to be enabled. This parameter can be one of the following values: <code>TIM_CHANNEL_1 / IM_CHANNEL_2</code></li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.13.4 HAL\_TIMEx\_OnePulseN\_Stop\_IT

Function Name	<b><i>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (</i></b> <b><i>TIM_HandleTypeDef * htim, uint32_t OutputChannel)</i></b>
Function Description	Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> <li>• <b>OutputChannel</b> : TIM Channel to be disabled. This parameter can be one of the following values: <code>TIM_CHANNEL_1 / IM_CHANNEL_2</code></li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 47.2.14 Peripheral Control functions

#### 47.2.14.1 HAL\_TIMEx\_ConfigCommutationEvent

Function Name	<b><i>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (</i></b> <b><i>TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)</i></b>
Function Description	Configure the TIM commutation event sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.</li> </ul>

- **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 / TIM\_TS\_ITR1 / TIM\_TS\_ITR2 / TIM\_TS\_ITR3 / TIM\_TS\_NONE
  - **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.

#### 47.2.14.2 HAL\_TIMEx\_ConfigCommutationEvent\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT ( <i>TIM_HandleTypeDef</i> * htim, uint32_t InputTrigger, uint32_t CommutationSource)</code>
Function Description	Configure the TIM commutation event sequence with interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> <li>• <b>InputTrigger</b> : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values: TIM_TS_ITR0 / TIM_TS_ITR1 / TIM_TS_ITR2 / TIM_TS_ITR3 / TIM_TS_NONE</li> <li>• <b>CommutationSource</b> : the Commutation Event source. This parameter can be one of the following values:           <ul style="list-style-type: none"> <li>– <b><i>TIM_COMMUTATION_TRGI</i></b> : Commutation source is the TRGI of the Interface Timer</li> <li>– <b><i>TIM_COMMUTATION_SOFTWARE</i></b> : Commutation source is set by software using the COMG bit</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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</li> </ul>

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.

#### 47.2.14.3 HAL\_TIMEx\_ConfigCommutationEvent\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_DMA (</b> <b>TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)</b>
Function Description	Configure the TIM commutation event sequence with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li><b>InputTrigger</b> : the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values: TIM_TS_ITR0 / TIM_TS_ITR1 / TIM_TS_ITR2 / TIM_TS_ITR3 / TIM_TS_NONE</li> <li><b>CommutationSource</b> : the Commutation Event source. This parameter can be one of the following values: <ul style="list-style-type: none"> <li><b>TIM_COMMUTATION_TRGI</b> : Commutation source is the TRGI of the Interface Timer</li> <li><b>TIM_COMMUTATION_SOFTWARE</b> : Commutation source is set by software using the COMG bit</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>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.</li> <li>: The user should configure the DMA in his own software, in This function only the COMDE bit is set</li> </ul>

#### 47.2.14.4 HAL\_TIMEx\_MasterConfigSynchronization

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (</b>
---------------	--

***TIM\_HandleTypeDef \* htim, TIM\_MasterConfigTypeDef \*  
sMasterConfig)***

Function Description	Configures the TIM in master mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>sMasterConfig</b> : pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.14.5 HAL\_TIMEx\_ConfigBreakDeadTime

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (</b> <b><i>TIM_HandleTypeDef * htim,</i></b> <b><i>TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)</i></b>
Function Description	Configures the Break feature, dead time, Lock level, OSS1/OSSR State and the AOE(automatic output enable).
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>sBreakDeadTimeConfig</b> : pointer to a TIM_BreakDeadTimeConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 47.2.14.6 HAL\_TIMEx\_RemapConfig

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_RemapConfig (</b> <b><i>TIM_HandleTypeDef * htim, uint32_t Remap)</i></b>
Function Description	Configures the TIM2, TIM5 and TIM11 Remapping input capabilities.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module..</li> </ul>

- **TIM\_Remap** : specifies the TIM input remapping source.  
This parameter can be one of the following values:
  - **TIM\_TIM2\_TIM8\_TRGO** : TIM2 ITR1 input is connected to TIM8 Trigger output(default)
  - **TIM\_TIM2\_ETH\_PTP** : TIM2 ITR1 input is connected to ETH PTP trigger output.
  - **TIM\_TIM2\_USBFS\_SOF** : TIM2 ITR1 input is connected to USB FS SOF.
  - **TIM\_TIM2\_USBHS\_SOF** : TIM2 ITR1 input is connected to USB HS SOF.
  - **TIM\_TIM5\_GPIO** : TIM5 CH4 input is connected to dedicated Timer pin(default)
  - **TIM\_TIM5\_LSI** : TIM5 CH4 input is connected to LSI clock.
  - **TIM\_TIM5\_LSE** : TIM5 CH4 input is connected to LSE clock.
  - **TIM\_TIM5\_RTC** : TIM5 CH4 input is connected to RTC Output event.
  - **TIM\_TIM11\_GPIO** : TIM11 CH4 input is connected to dedicated Timer pin(default)
  - **TIM\_TIM11\_HSE** : TIM11 CH4 input is connected to HSE\_RTC clock (HSE divided by a programmable prescaler)

Return values

- **HAL status**

Notes

- None.

## 47.2.15 Extension Callbacks functions

### 47.2.15.1 HAL\_TIMEx\_CommmutationCallback

Function Name	<b>void HAL_TIMEx_CommmutationCallback ( <i>TIM_HandleTypeDef</i> * <i>htim</i> )</b>
Function Description	Hall commutation changed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 47.2.15.2 HAL\_TIMEx\_BreakCallback

Function Name	<b>void HAL_TIMEx_BreakCallback ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Hall Break detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 47.2.16 Extension Peripheral State functions

#### 47.2.16.1 HAL\_TIMEx\_HallSensor\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState ( <i>TIM_HandleTypeDef</i> * htim)</b>
Function Description	Return the TIM Hall Sensor interface state.
Parameters	<ul style="list-style-type: none"><li>• <b>htim</b> : pointer to a <i>TIM_HandleTypeDef</i> structure that contains the configuration information for TIM module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

## 47.3 TIMEx Firmware driver defines

### 47.3.1 TIMEx

TIMEx

*TIMEx\_AOE\_Bit\_Set\_Reset*

- #define: ***TIM\_AUTOMATICOUTPUT\_ENABLE (TIM\_BDTR\_AOE)***
- #define: ***TIM\_AUTOMATICOUTPUT\_DISABLE ((uint32\_t)0x0000)***

*TIMEx\_Break\_Input\_enable\_disable*

- #define: ***TIM\_BREAK\_ENABLE (TIM\_BDTR\_BKE)***
  
- #define: ***TIM\_BREAK\_DISABLE ((uint32\_t)0x0000)***

*TIMEx\_Break\_Polarity*

- #define: ***TIM\_BREAKPOLARITY\_LOW ((uint32\_t)0x0000)***
  
- #define: ***TIM\_BREAKPOLARITY\_HIGH (TIM\_BDTR\_BKP)***

*TIMEx\_Commputation\_Mode*

- #define: ***TIM\_COMMUTATION\_TRGI (TIM\_CR2\_CCUS)***
  
- #define: ***TIM\_COMMUTATION\_SOFTWARE ((uint32\_t)0x0000)***

*TIMEx\_Lock\_level*

- #define: ***TIM\_LOCKLEVEL\_OFF ((uint32\_t)0x0000)***
  
- #define: ***TIM\_LOCKLEVEL\_1 (TIM\_BDTR\_LOCK\_0)***
  
- #define: ***TIM\_LOCKLEVEL\_2 (TIM\_BDTR\_LOCK\_1)***
  
- #define: ***TIM\_LOCKLEVEL\_3 (TIM\_BDTR\_LOCK)***

*TIMEx\_Master\_Mode\_Selection*

- #define: ***TIM\_TRGO\_RESET ((uint32\_t)0x0000)***

- #define: ***TIM\_TRGO\_ENABLE (TIM\_CR2\_MMS\_0)***
- #define: ***TIM\_TRGO\_UPDATE (TIM\_CR2\_MMS\_1)***
- #define: ***TIM\_TRGO\_OC1 ((TIM\_CR2\_MMS\_1 | TIM\_CR2\_MMS\_0))***
- #define: ***TIM\_TRGO\_OC1REF (TIM\_CR2\_MMS\_2)***
- #define: ***TIM\_TRGO\_OC2REF ((TIM\_CR2\_MMS\_2 | TIM\_CR2\_MMS\_0))***
- #define: ***TIM\_TRGO\_OC3REF ((TIM\_CR2\_MMS\_2 | TIM\_CR2\_MMS\_1))***
- #define: ***TIM\_TRGO\_OC4REF ((TIM\_CR2\_MMS\_2 | TIM\_CR2\_MMS\_1 | TIM\_CR2\_MMS\_0))***

#### ***TIMEx\_Master\_Slave\_Mode***

- #define: ***TIM\_MASTERSLAVEMODE\_ENABLE ((uint32\_t)0x0080)***
- #define: ***TIM\_MASTERSLAVEMODE\_DISABLE ((uint32\_t)0x0000)***

#### ***TIMEx\_OSSI\_Off\_State\_Selection\_for\_Idle\_mode\_state***

- #define: ***TIM\_OSSI\_ENABLE (TIM\_BDTR\_OSSI)***
- #define: ***TIM\_OSSI\_DISABLE ((uint32\_t)0x0000)***

***TIMEx\_OSSR\_Off\_State\_Selection\_for\_Run\_mode\_state***

- #define: ***TIM\_OSSR\_ENABLE (TIM\_BDTR\_OSSR)***
  
- #define: ***TIM\_OSSR\_DISABLE ((uint32\_t)0x0000)***

***TIMEx\_Remap***

- #define: ***TIM\_TIM2\_TIM8\_TRGO (0x00000000)***
  
- #define: ***TIM\_TIM2\_ETH\_PTP (0x00000400)***
  
- #define: ***TIM\_TIM2\_USBFS\_SOF (0x00000800)***
  
- #define: ***TIM\_TIM2\_USBHS\_SOF (0x00000C00)***
  
- #define: ***TIM\_TIM5\_GPIO (0x00000000)***
  
- #define: ***TIM\_TIM5\_LSI (0x00000040)***
  
- #define: ***TIM\_TIM5\_LSE (0x00000080)***
  
- #define: ***TIM\_TIM5\_RTC (0x000000C0)***
  
- #define: ***TIM\_TIM11\_GPIO (0x00000000)***

- #define: **TIM\_TIM11\_HSE (0x00000002)**

## 48 HAL UART Generic Driver

### 48.1 UART Firmware driver registers structures

#### 48.1.1 **UART\_HandleTypeDef**

**UART\_HandleTypeDef** is defined in the `stm32f4xx_hal_uart.h`

##### Data Fields

- **`USART_TypeDef * Instance`**
- **`UART_InitTypeDef Init`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferSize`**
- **`uint16_t TxXferCount`**
- **`uint8_t * pRxBuffPtr`**
- **`uint16_t RxXferSize`**
- **`uint16_t RxXferCount`**
- **`DMA_HandleTypeDef * hdmatx`**
- **`DMA_HandleTypeDef * hdmarx`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_UART_StateTypeDef State`**
- **`__IO HAL_UART_ErrorTypeDef ErrorCode`**

##### Field Documentation

- **`USART_TypeDef* UART_HandleTypeDef::Instance`**
- **`UART_InitTypeDef UART_HandleTypeDef::Init`**
- **`uint8_t* UART_HandleTypeDef::pTxBuffPtr`**
- **`uint16_t UART_HandleTypeDef::TxXferSize`**
- **`uint16_t UART_HandleTypeDef::TxXferCount`**
- **`uint8_t* UART_HandleTypeDef::pRxBuffPtr`**
- **`uint16_t UART_HandleTypeDef::RxXferSize`**
- **`uint16_t UART_HandleTypeDef::RxXferCount`**
- **`DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx`**
- **`DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx`**
- **`HAL_LockTypeDef UART_HandleTypeDef::Lock`**
- **`__IO HAL_UART_StateTypeDef UART_HandleTypeDef::State`**
- **`__IO HAL_UART_ErrorTypeDef UART_HandleTypeDef::ErrorCode`**

#### 48.1.2 **UART\_InitTypeDef**

**UART\_InitTypeDef** is defined in the `stm32f4xx_hal_uart.h`

##### Data Fields

- **`uint32_t BaudRate`**

- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t HwFlowCtl*
- *uint32\_t OverSampling*

### Field Documentation

- *uint32\_t UART\_InitTypeDef::BaudRate*
  - This member configures the UART communication baud rate. The baud rate is computed using the following formula: IntegerDivider = ((PCLKx) / (8 \* (OVR8+1) \* (huart->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32\_t) IntegerDivider)) \* 8 \* (OVR8+1)) + 0.5 Where OVR8 is the "oversampling by 8 mode" configuration bit in the CR1 register.
- *uint32\_t UART\_InitTypeDef::WordLength*
  - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [UART\\_Word\\_Length](#)
- *uint32\_t UART\_InitTypeDef::StopBits*
  - Specifies the number of stop bits transmitted. This parameter can be a value of [UART\\_Stop\\_Bits](#)
- *uint32\_t UART\_InitTypeDef::Parity*
  - Specifies the parity mode. This parameter can be a value of [UART\\_Parity](#)
- *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](#)

### 48.1.3 USART\_TypeDef

**USART\_TypeDef** is defined in the stm32f439xx.h

#### Data Fields

- *\_\_IO uint32\_t SR*
- *\_\_IO uint32\_t DR*
- *\_\_IO uint32\_t BRR*
- *\_\_IO uint32\_t CR1*
- *\_\_IO uint32\_t CR2*
- *\_\_IO uint32\_t CR3*
- *\_\_IO uint32\_t GTPR*

### Field Documentation

- **`_IO uint32_t USART_TypeDef::SR`**
  - USART Status register, Address offset: 0x00
- **`_IO uint32_t USART_TypeDef::DR`**
  - USART Data register, Address offset: 0x04
- **`_IO uint32_t USART_TypeDef::BRR`**
  - USART Baud rate register, Address offset: 0x08
- **`_IO uint32_t USART_TypeDef::CR1`**
  - USART Control register 1, Address offset: 0x0C
- **`_IO uint32_t USART_TypeDef::CR2`**
  - USART Control register 2, Address offset: 0x10
- **`_IO uint32_t USART_TypeDef::CR3`**
  - USART Control register 3, Address offset: 0x14
- **`_IO uint32_t USART_TypeDef::GTPR`**
  - USART Guard time and prescaler register, Address offset: 0x18

## 48.2 UART Firmware driver API description

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

### 48.2.1 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 stream.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.
    - Configure the DMA Tx/Rx Stream.
    - Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the Init structure.
4. For the UART asynchronous mode, initialize the UART registers by calling the `HAL_UART_Init()` API.
5. For the UART Half duplex mode, initialize the UART registers by calling the `HAL_HalfDuplex_Init()` API.
6. For the LIN mode, initialize the UART registers by calling the `HAL_LIN_Init()` API.

7. For the Multi-Processor mode, initialize the UART registers by calling the HAL\_MultiProcessor\_Init() API.



The specific UART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros \_\_HAL\_UART\_ENABLE\_IT() and \_\_HAL\_UART\_DISABLE\_IT() inside the transmit and receive process.



These APIs (HAL\_UART\_Init() and HAL\_HalfDuplex\_Init()) configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the 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\_DMASStop()

### 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

## 48.2.2 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. Refer to the STM32F4xx reference manual (RM0090) for the UART frame formats depending on the frame length defined by the M bit (8-bits or 9-bits).
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Methode

The HAL\_UART\_Init(), HAL\_HalfDuplex\_Init(), HAL\_LIN\_Init() and HAL\_MultiProcessor\_Init() APIs follow respectively the UART asynchronous, UART Half duplex, LIN and Multi-Processor configuration procedures (details for the procedures are available in reference manual (RM0329)).

- [`HAL\_UART\_Init\(\)`](#)
- [`HAL\_HalfDuplex\_Init\(\)`](#)
- [`HAL\_LIN\_Init\(\)`](#)
- [`HAL\_MultiProcessor\_Init\(\)`](#)
- [`HAL\_UART\_DelInit\(\)`](#)
- [`HAL\_UART\_MspInit\(\)`](#)

- [\*HAL\\_UART\\_MspDeInit\(\)\*](#)

### 48.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the UART asynchronous and Half duplex data transfers.

1. There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - Non blocking mode: The communication is performed using Interrupts or DMA, these APIs return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The *HAL\_UART\_TxCpltCallback()*, *HAL\_UART\_RxCpltCallback()* user callbacks will be executed respectively at the end of the transmit or receive process. The *HAL\_UART\_ErrorCallback()* user callback will be executed when a communication error is detected.
2. Blocking mode APIs are:
  - *HAL\_UART\_Transmit()*
  - *HAL\_UART\_Receive()*
3. Non Blocking mode APIs with Interrupt are:
  - *HAL\_UART\_Transmit\_IT()*
  - *HAL\_UART\_Receive\_IT()*
  - *HAL\_UART\_IRQHandler()*
4. Non Blocking mode functions with DMA are:
  - *HAL\_UART\_Transmit\_DMA()*
  - *HAL\_UART\_Receive\_DMA()*
5. A set of Transfer Complete Callbacks are provided in non blocking mode:
  - *HAL\_UART\_TxCpltCallback()*
  - *HAL\_UART\_RxCpltCallback()*
  - *HAL\_UART\_ErrorCallback()*



In the Half duplex communication, it is forbidden to run the transmit and receive process in parallel, the UART state *HAL\_UART\_STATE\_BUSY\_TX\_RX* can't be useful.

- [\*HAL\\_UART\\_Transmit\(\)\*](#)
- [\*HAL\\_UART\\_Receive\(\)\*](#)
- [\*HAL\\_UART\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_UART\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_UART\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_UART\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_UART\\_DMAPause\(\)\*](#)
- [\*HAL\\_UART\\_DMAResume\(\)\*](#)
- [\*HAL\\_UART\\_DMAStop\(\)\*](#)
- [\*HAL\\_UART\\_IRQHandler\(\)\*](#)
- [\*HAL\\_UART\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_TxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_ErrorCallback\(\)\*](#)

#### 48.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART:

- HAL\_LIN\_SendBreak() API can be helpful to transmit the break character.
- HAL\_MultiProcessor\_EnterMuteMode() API can be helpful to enter the UART in mute mode.
- HAL\_MultiProcessor\_ExitMuteMode() API can be helpful to exit the UART mute mode by software.
- [\*\*HAL\\_LIN\\_SendBreak\(\)\*\*](#)
- [\*\*HAL\\_MultiProcessor\\_EnterMuteMode\(\)\*\*](#)
- [\*\*HAL\\_MultiProcessor\\_ExitMuteMode\(\)\*\*](#)
- [\*\*HAL\\_HalfDuplex\\_EnableTransmitter\(\)\*\*](#)
- [\*\*HAL\\_HalfDuplex\\_EnableReceiver\(\)\*\*](#)

#### 48.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of UART communication process, return Peripheral Errors occurred during communication process

- HAL\_UART\_GetState() API can be helpful to check in run-time the state of the UART peripheral.
- HAL\_UART\_GetError() check in run-time errors that could be occurred during communication.
- [\*\*HAL\\_UART\\_GetState\(\)\*\*](#)
- [\*\*HAL\\_UART\\_GetError\(\)\*\*](#)

#### 48.2.6 Initialization and de-initialization functions

##### 48.2.6.1 HAL\_UART\_Init

Function Name	<b>HAL_StatusTypeDef HAL_UART_Init ( <a href="#">UART_HandleTypeDef * huart</a> )</b>
Function Description	Initializes the UART mode according to the specified parameters in the <code>UART_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <code>UART_HandleTypeDef</code> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

##### 48.2.6.2 HAL\_HalfDuplex\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_Init (</b> <b><i>UART_HandleTypeDef * huart</i></b> )
Function Description	Initializes the half-duplex mode according to the specified parameters in the <b>UART_InitTypeDef</b> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <b>UART_HandleTypeDef</b> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.6.3 HAL\_LIN\_Init

Function Name	<b>HAL_StatusTypeDef HAL_LIN_Init (</b> <b><i>UART_HandleTypeDef * huart, uint32_t BreakDetectLength</i></b> )
Function Description	Initializes the LIN mode according to the specified parameters in the <b>UART_InitTypeDef</b> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <b>UART_HandleTypeDef</b> structure that contains the configuration information for the specified UART module.</li> <li>• <b>BreakDetectLength</b> : Specifies the LIN break detection length. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <b><i>UART_LINBREAKDETECTLENGTH_10B</i></b> : 10-bit break detection</li> <li>– <b><i>UART_LINBREAKDETECTLENGTH_11B</i></b> : 11-bit break detection</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.6.4 HAL\_MultiProcessor\_Init

Function Name	<b>HAL_StatusTypeDef HAL_MultiProcessor_Init (</b> <b><i>UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethode</i></b> )
---------------	---

Function Description	Initializes the Multi-Processor mode according to the specified parameters in the <code>UART_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <code>UART_HandleTypeDef</code> structure that contains the configuration information for the specified UART module.</li> <li>• <b>Address</b> : USART address</li> <li>• <b>WakeUpMethode</b> : specifies the USART wakeup method. This parameter can be one of the following values: <ul style="list-style-type: none"> <li>– <code>UART_WAKEUPMETHODE_IDLELINE</code> : Wakeup by an idle line detection</li> <li>– <code>UART_WAKEUPMETHODE_ADDRESSMARK</code> : Wakeup by an address mark</li> </ul> </li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.6.5 HAL\_UART\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_UART_DeInit (</code> <code>UART_HandleTypeDef * huart)</code>
Function Description	Deinitializes the USART peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <code>UART_HandleTypeDef</code> structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.6.6 HAL\_UART\_MspInit

Function Name	<code>void HAL_UART_MspInit (</code> <code>UART_HandleTypeDef * huart)</code>
Function Description	USART MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <code>UART_HandleTypeDef</code> structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## Notes

- None.

#### 48.2.6.7 HAL\_UART\_MspDeInit

Function Name	<b>void HAL_UART_MspDeInit ( <i>UART_HandleTypeDef</i> * huart)</b>
Function Description	UART MSP DeInit.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

### 48.2.7 IO operation functions

#### 48.2.7.1 HAL\_UART\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit ( <i>UART_HandleTypeDef</i> * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Sends an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li><li>• <b>pData</b> : Pointer to data buffer</li><li>• <b>Size</b> : Amount of data to be sent</li><li>• <b>Timeout</b> : Timeout duration</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 48.2.7.2 HAL\_UART\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive (</b> <b>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size,</b> <b>uint32_t Timeout)</b>
Function Description	Receives an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 48.2.7.3 HAL\_UART\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_IT (</b> <b>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Sends an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 48.2.7.4 HAL\_UART\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_IT (</b> <b>UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Receives an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> </ul>

- **pData** : Pointer to data buffer
  - **Size** : Amount of data to be received
- Return values
- **HAL status**
- Notes
- None.

#### 48.2.7.5 HAL\_UART\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_DMA (</b> <b><i>UART_HandleTypeDef</i> * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Sends an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	• <b>HAL status</b>
Notes	• None.

#### 48.2.7.6 HAL\_UART\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_DMA (</b> <b><i>UART_HandleTypeDef</i> * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Receives an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> <li>• <b>pData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> </ul>
Return values	• <b>HAL status</b>
Notes	• When the UART parity is enabled (PCE = 1) the data received contain the parity bit.

---

#### 48.2.7.7 HAL\_UART\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAPause (</b> <i>UART_HandleTypeDef * huart)</i>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.7.8 HAL\_UART\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAResume (</b> <i>UART_HandleTypeDef * huart)</i>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.7.9 HAL\_UART\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAStop (</b> <i>UART_HandleTypeDef * huart)</i>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

#### 48.2.7.10 HAL\_UART\_IRQHandler

Function Name	<b>void HAL_UART_IRQHandler ( <i>UART_HandleTypeDef</i> * huart)</b>
Function Description	This function handles UART interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 48.2.7.11 HAL\_UART\_TxCpltCallback

Function Name	<b>void HAL_UART_TxCpltCallback ( <i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 48.2.7.12 HAL\_UART\_TxHalfCpltCallback

Function Name	<b>void HAL_UART_TxHalfCpltCallback ( <i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Tx Half Transfer completed callbacks.

Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : pointer to a <code>UART_HandleTypeDef</code> structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 48.2.7.13 HAL\_UART\_RxCpltCallback

Function Name	<code>void HAL_UART_RxCpltCallback ( <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : pointer to a <code>UART_HandleTypeDef</code> structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 48.2.7.14 HAL\_UART\_RxHalfCpltCallback

Function Name	<code>void HAL_UART_RxHalfCpltCallback ( <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>huart</b> : pointer to a <code>UART_HandleTypeDef</code> structure that contains the configuration information for the specified UART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 48.2.7.15 HAL\_UART\_ErrorCallback

Function Name	<code>void HAL_UART_ErrorCallback ( <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	UART error callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 48.2.8 Peripheral Control functions

### 48.2.8.1 HAL\_LIN\_SendBreak

Function Name	<code>HAL_StatusTypeDef HAL_LIN_SendBreak ( <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	Transmits break characters.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 48.2.8.2 HAL\_MultiProcessor\_EnterMuteMode

Function Name	<code>HAL_StatusTypeDef HAL_MultiProcessor_EnterMuteMode ( <i>UART_HandleTypeDef</i> * huart)</code>
Function Description	Enters the UART in mute mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.8.3 HAL\_MultiProcessor\_ExitMuteMode

Function Name	<b>HAL_StatusTypeDef HAL_MultiProcessor_ExitMuteMode (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Exits the UART mute mode: wake up software.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.8.4 HAL\_HalfDuplex\_EnableTransmitter

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Enables the UART transmitter and disables the UART receiver.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 48.2.8.5 HAL\_HalfDuplex\_EnableReceiver

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (</b> <b><i>UART_HandleTypeDef</i> * huart)</b>
Function Description	Enables the UART receiver and disables the UART transmitter.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart</b> : pointer to a <i>UART_HandleTypeDef</i> structure that</li> </ul>

contains the configuration information for the specified UART module.

- |               |                     |
|---------------|---------------------|
| Return values | • <b>HAL status</b> |
| Notes         | • None.             |

## 48.2.9 Peripheral State and Errors functions

### 48.2.9.1 HAL\_UART\_GetState

Function Name	<b>HAL_UART_StateTypeDef HAL_UART_GetState (</b> <b><i>UART_HandleTypeDef * huart</i></b> )
Function Description	Returns the UART state.
Parameters	• <b>huart</b> : pointer to a <b>UART_HandleTypeDef</b> structure that contains the configuration information for the specified UART module.
Return values	• <b>HAL state</b>
Notes	• None.

### 48.2.9.2 HAL\_UART\_GetError

Function Name	<b>uint32_t HAL_UART_GetError (</b> <b><i>UART_HandleTypeDef * huart</i></b> )
Function Description	Return the UART error code.
Parameters	• <b>huart</b> : pointer to a <b>UART_HandleTypeDef</b> structure that contains the configuration information for the specified UART.
Return values	• <b>UART Error Code</b>
Notes	• None.

## 48.3 UART Firmware driver defines

### 48.3.1 UART

UART

#### *UART\_Flags*

- #define: ***UART\_FLAG\_CTS*** ((*uint32\_t*)0x000000200)
- #define: ***UART\_FLAG\_LBD*** ((*uint32\_t*)0x000000100)
- #define: ***UART\_FLAG\_TXE*** ((*uint32\_t*)0x000000080)
- #define: ***UART\_FLAG\_TC*** ((*uint32\_t*)0x000000040)
- #define: ***UART\_FLAG\_RXNE*** ((*uint32\_t*)0x000000020)
- #define: ***UART\_FLAG\_IDLE*** ((*uint32\_t*)0x000000010)
- #define: ***UART\_FLAG\_ORE*** ((*uint32\_t*)0x000000008)
- #define: ***UART\_FLAG\_NE*** ((*uint32\_t*)0x000000004)
- #define: ***UART\_FLAG\_FE*** ((*uint32\_t*)0x000000002)
- #define: ***UART\_FLAG\_PE*** ((*uint32\_t*)0x000000001)

#### *UART\_Hardware\_Flow\_Control*

- #define: ***UART\_HWCONTROL\_NONE*** ((*uint32\_t*)0x00000000)

- #define: ***UART\_HWCONTROL\_RTS*** ((*uint32\_t*)***USART\_CR3\_RTSE***)
- #define: ***UART\_HWCONTROL\_CTS*** ((*uint32\_t*)***USART\_CR3\_CTSE***)
- #define: ***UART\_HWCONTROL\_RTS\_CTS*** ((*uint32\_t*)(***USART\_CR3\_RTSE | USART\_CR3\_CTSE***))

#### ***UART\_Interrupt\_definition***

- #define: ***UART\_IT\_PE*** ((*uint32\_t*)0x10000100)
- #define: ***UART\_IT\_TXE*** ((*uint32\_t*)0x10000080)
- #define: ***UART\_IT\_TC*** ((*uint32\_t*)0x10000040)
- #define: ***UART\_IT\_RXNE*** ((*uint32\_t*)0x10000020)
- #define: ***UART\_IT\_IDLE*** ((*uint32\_t*)0x10000010)
- #define: ***UART\_IT\_LBD*** ((*uint32\_t*)0x20000040)
- #define: ***UART\_IT\_CTS*** ((*uint32\_t*)0x30000400)
- #define: ***UART\_IT\_ERR*** ((*uint32\_t*)0x30000001)

***UART\_LIN\_Break\_Detection\_Length***

- #define: ***UART\_LINBREAKDETECTLENGTH\_10B*** ((*uint32\_t*)0x00000000)
  
- #define: ***UART\_LINBREAKDETECTLENGTH\_11B*** ((*uint32\_t*)0x00000020)

***UART\_Mode***

- #define: ***UART\_MODE\_RX*** ((*uint32\_t*)***USART\_CR1\_RE***)
  
- #define: ***UART\_MODE\_TX*** ((*uint32\_t*)***USART\_CR1\_TE***)
  
- #define: ***UART\_MODE\_TX\_RX*** ((*uint32\_t*)(***USART\_CR1\_TE | USART\_CR1\_RE***))

***UART\_Over\_Sampling***

- #define: ***UART\_OVERSAMPLING\_16*** ((*uint32\_t*)0x00000000)
  
- #define: ***UART\_OVERSAMPLING\_8*** ((*uint32\_t*)***USART\_CR1\_OVER8***)

***UART\_Parity***

- #define: ***UART\_PARITY\_NONE*** ((*uint32\_t*)0x00000000)
  
- #define: ***UART\_PARITY\_EVEN*** ((*uint32\_t*)***USART\_CR1\_PCE***)
  
- #define: ***UART\_PARITY\_ODD*** ((*uint32\_t*)(***USART\_CR1\_PCE | USART\_CR1\_PS***))

***UART\_State***

- #define: ***UART\_STATE\_DISABLE*** ((*uint32\_t*)0x00000000)

- #define: **UART\_STATE\_ENABLE** ((*uint32\_t*)USART\_CR1\_UE)

#### ***UART\_Stop\_Bits***

- #define: **UART\_STOPBITS\_1** ((*uint32\_t*)0x00000000)
- #define: **UART\_STOPBITS\_2** ((*uint32\_t*)USART\_CR2\_STOP\_1)

#### ***UART\_WakeUp\_functions***

- #define: **UART\_WAKEUPMETHODE\_IDLELINE** ((*uint32\_t*)0x00000000)
- #define: **UART\_WAKEUPMETHODE\_ADDRESSMARK** ((*uint32\_t*)0x00000800)

#### ***UART\_Word\_Length***

- #define: **UART\_WORDLENGTH\_8B** ((*uint32\_t*)0x00000000)
- #define: **UART\_WORDLENGTH\_9B** ((*uint32\_t*)USART\_CR1\_M)

## 49 HAL USART Generic Driver

### 49.1 USART Firmware driver registers structures

#### 49.1.1 USART\_HandleTypeDef

*USART\_HandleTypeDef* is defined in the `stm32f4xx_hal_usart.h`

##### Data Fields

- *USART\_TypeDef \* Instance*
- *USART\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *uint16\_t TxXferSize*
- *\_IO uint16\_t TxXferCount*
- *uint8\_t \* pRxBuffPtr*
- *uint16\_t RxXferSize*
- *\_IO uint16\_t RxXferCount*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *HAL\_LockTypeDef Lock*
- *\_IO HAL\_USART\_StateTypeDef State*
- *\_IO HAL\_USART\_ErrorTypeDef ErrorCode*

##### Field Documentation

- *USART\_TypeDef\* USART\_HandleTypeDef::Instance*
- *USART\_InitTypeDef USART\_HandleTypeDef::Init*
- *uint8\_t\* USART\_HandleTypeDef::pTxBuffPtr*
- *uint16\_t USART\_HandleTypeDef::TxXferSize*
- *\_IO uint16\_t USART\_HandleTypeDef::TxXferCount*
- *uint8\_t\* USART\_HandleTypeDef::pRxBuffPtr*
- *uint16\_t USART\_HandleTypeDef::RxXferSize*
- *\_IO uint16\_t USART\_HandleTypeDef::RxXferCount*
- *DMA\_HandleTypeDef\* USART\_HandleTypeDef::hdmatx*
- *DMA\_HandleTypeDef\* USART\_HandleTypeDef::hdmarx*
- *HAL\_LockTypeDef USART\_HandleTypeDef::Lock*
- *\_IO HAL\_USART\_StateTypeDef USART\_HandleTypeDef::State*
- *\_IO HAL\_USART\_ErrorTypeDef USART\_HandleTypeDef::ErrorCode*

#### 49.1.2 USART\_InitTypeDef

*USART\_InitTypeDef* is defined in the `stm32f4xx_hal_usart.h`

##### Data Fields

- *uint32\_t BaudRate*

- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t CLKLastBit*

### Field Documentation

- ***uint32\_t USART\_InitTypeDef::BaudRate***
  - This member configures the Usart communication baud rate. The baud rate is computed using the following formula: IntegerDivider = ((PCLKx) / (8 \* (husart->Init.BaudRate)))FractionalDivider = ((IntegerDivider - ((uint32\_t) IntegerDivider)) \* 8) + 0.5
- ***uint32\_t USART\_InitTypeDef::WordLength***
  - Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART\_Word\_Length***
- ***uint32\_t USART\_InitTypeDef::StopBits***
  - Specifies the number of stop bits transmitted. This parameter can be a value of ***USART\_Stop\_Bits***
- ***uint32\_t USART\_InitTypeDef::Parity***
  - Specifies the parity mode. This parameter can be a value of ***USART\_Parity***
- ***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***

### 49.1.3 USART\_TypeDef

***USART\_TypeDef*** is defined in the stm32f439xx.h

#### Data Fields

- *\_\_IO uint32\_t SR*
- *\_\_IO uint32\_t DR*
- *\_\_IO uint32\_t BRR*
- *\_\_IO uint32\_t CR1*
- *\_\_IO uint32\_t CR2*
- *\_\_IO uint32\_t CR3*
- *\_\_IO uint32\_t GTPR*

## Field Documentation

- `__IO uint32_t USART_TypeDef::SR`  
– USART Status register, Address offset: 0x00
- `__IO uint32_t USART_TypeDef::DR`  
– USART Data register, Address offset: 0x04
- `__IO uint32_t USART_TypeDef::BRR`  
– USART Baud rate register, Address offset: 0x08
- `__IO uint32_t USART_TypeDef::CR1`  
– USART Control register 1, Address offset: 0x0C
- `__IO uint32_t USART_TypeDef::CR2`  
– USART Control register 2, Address offset: 0x10
- `__IO uint32_t USART_TypeDef::CR3`  
– USART Control register 3, Address offset: 0x14
- `__IO uint32_t USART_TypeDef::GTPR`  
– USART Guard time and prescaler register, Address offset: 0x18

## 49.2 USART Firmware driver API description

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

### 49.2.1 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\_IT() and HAL\_USART\_TransmitReceive\_IT() APIs):
    - Declare a DMA handle structure for the Tx/Rx stream.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.
    - Configure the DMA Tx/Rx Stream.
    - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.

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 \_\_USART\_ENABLE\_IT() and \_\_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

## 49.2.2 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. Refer to the STM32F4xx reference manual (RM0090) for the USART frame formats depending on the frame length defined by the M bit (8-bits or 9-bits).
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes

The HAL\_USART\_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual (RM0329)).

- [\*\*HAL\\_USART\\_Init\(\)\*\*](#)
- [\*\*HAL\\_USART\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_MspDelInit\(\)\*\*](#)

## 49.2.3 IO operation functions

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

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_USART\_TxCpltCallback(), HAL\_USART\_RxCpltCallback() and HAL\_USART\_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process. The HAL\_USART\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
  - HAL\_USART\_Transmit() in simplex mode
  - HAL\_USART\_Receive() in full duplex receive only
  - HAL\_USART\_TransmitReceive() in full duplex mode
3. Non Blocking mode APIs with Interrupt are :
  - HAL\_USART\_Transmit\_IT() in simplex mode
  - HAL\_USART\_Receive\_IT() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_IT() in full duplex mode
  - HAL\_USART\_IRQHandler()
4. Non Blocking mode functions with DMA are :
  - HAL\_USART\_Transmit\_DMA() in simplex mode
  - HAL\_USART\_Receive\_DMA() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_DMA() in full duplex mode
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_USART\_TxCpltCallback()
  - HAL\_USART\_RxCpltCallback()
  - HAL\_USART\_ErrorCallback()
  - HAL\_USART\_TxRxCpltCallback()
  - **HAL\_USART\_Transmit()**
  - **HAL\_USART\_Receive()**
  - **HAL\_USART\_TransmitReceive()**
  - **HAL\_USART\_Transmit\_IT()**
  - **HAL\_USART\_Receive\_IT()**
  - **HAL\_USART\_TransmitReceive\_IT()**
  - **HAL\_USART\_Transmit\_DMA()**
  - **HAL\_USART\_Receive\_DMA()**
  - **HAL\_USART\_TransmitReceive\_DMA()**
  - **HAL\_USART\_DMAPause()**
  - **HAL\_USART\_DMAResume()**
  - **HAL\_USART\_DMAStop()**
  - **HAL\_USART\_IRQHandler()**
  - **HAL\_USART\_TxCpltCallback()**
  - **HAL\_USART\_TxHalfCpltCallback()**
  - **HAL\_USART\_RxCpltCallback()**
  - **HAL\_USART\_RxHalfCpltCallback()**
  - **HAL\_USART\_TxRxCpltCallback()**
  - **HAL\_USART\_ErrorCallback()**

## 49.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of USART communication process, return Peripheral Errors occurred during communication process

- HAL\_USART\_GetState() API can be helpful to check in run-time the state of the USART peripheral.
- HAL\_USART\_GetError() check in run-time errors that could be occurred during communication.
- [\*\*HAL\\_USART\\_GetState\(\)\*\*](#)
- [\*\*HAL\\_USART\\_GetError\(\)\*\*](#)

## 49.2.5 USART Initialization and de-initialization functions

### 49.2.5.1 HAL\_USART\_Init

Function Name	<b>HAL_StatusTypeDef HAL_USART_Init (<a href="#">USART_HandleTypeDef * husart</a>)</b>
Function Description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 49.2.5.2 HAL\_USART\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_USART_DelInit (<a href="#">USART_HandleTypeDef * husart</a>)</b>
Function Description	Deinitializes the USART peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.5.3 HAL\_USART\_MspInit

Function Name	<b>void HAL_USART_MspInit ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	USART MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.5.4 HAL\_USART\_MspDeInit

Function Name	<b>void HAL_USART_MspDeInit ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	USART MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 49.2.6 IO operation functions

#### 49.2.6.1 HAL\_USART\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit ( <i>USART_HandleTypeDef</i> * <i>husart</i>, <i>uint8_t</i> * <i>pTxData</i>, <i>uint16_t</i> <i>Size</i>, <i>uint32_t</i> <i>Timeout</i>)</b>
Function Description	Simplex Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pTxData</b> : Pointer to data buffer</li> </ul>

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"> <li>• <b>Size</b> : Amount of data to be sent</li> <li>• <b>Timeout</b> : Timeout duration</li> <li>• <b>HAL status</b></li> </ul> |
| Notes         | <ul style="list-style-type: none"> <li>• None.</li> </ul>   |

#### 49.2.6.2 HAL\_USART\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive (</b> <b><i>USART_HandleTypeDef</i> * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Full-Duplex Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pRxData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.6.3 HAL\_USART\_TransmitReceive

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive (</b> <b><i>USART_HandleTypeDef</i> * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Full-Duplex Send receive an amount of data in full-duplex mode (blocking mode).
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pTxData</b> : Pointer to data transmitted buffer</li> <li>• <b>pRxData</b> : Pointer to data received buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> <li>• <b>Timeout</b> : Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

- |       |   |
|-------|---|
| Notes | <ul style="list-style-type: none"><li>• None.</li></ul> |
|-------|---|

#### 49.2.6.4 HAL\_USART\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit_IT (</b> <b>USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t</b> <b>Size)</b>
Function Description	Simplex Send an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li><li>• <b>pTxData</b> : Pointer to data buffer</li><li>• <b>Size</b> : Amount of data to be sent</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• The USART errors are not managed to avoid the overrun error.</li></ul>

#### 49.2.6.5 HAL\_USART\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive_IT (</b> <b>USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t</b> <b>Size)</b>
Function Description	Simplex Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li><li>• <b>pRxData</b> : Pointer to data buffer</li><li>• <b>Size</b> : Amount of data to be received</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL status</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 49.2.6.6 HAL\_USART\_TransmitReceive\_IT

Function Name	<code>HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (     <b>USART_HandleTypeDef</b> * husart, uint8_t * pTxData, uint8_t *     pRxData, uint16_t Size)</code>
Function Description	Full-Duplex Send receive an amount of data in full-duplex mode (non-blocking).
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pTxData</b> : Pointer to data transmitted buffer</li> <li>• <b>pRxData</b> : Pointer to data received buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.6.7 HAL\_USART\_Transmit\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_Transmit_DMA (     <b>USART_HandleTypeDef</b> * husart, uint8_t * pTxData, uint16_t     Size)</code>
Function Description	Simplex Send an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pTxData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.6.8 HAL\_USART\_Receive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_Receive_DMA (     <b>USART_HandleTypeDef</b> * husart, uint8_t * pRxData, uint16_t</code>
---------------	---

<b>Size)</b>	
Function Description	Full-Duplex Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pRxData</b> : Pointer to data buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> </ul>
Return values	• <b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• The USART DMA transmit stream must be configured in order to generate the clock for the slave.</li> <li>• When the USART parity is enabled (PCE = 1) the data received contain the parity bit.</li> </ul>

#### 49.2.6.9 HAL\_USART\_TransmitReceive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (</b> <b>USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
Function Description	Full-Duplex Transmit Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> <li>• <b>pTxData</b> : Pointer to data transmitted buffer</li> <li>• <b>pRxData</b> : Pointer to data received buffer</li> <li>• <b>Size</b> : Amount of data to be received</li> </ul>
Return values	• <b>HAL status</b>
Notes	<ul style="list-style-type: none"> <li>• When the USART parity is enabled (PCE = 1) the data received contain the parity bit.</li> </ul>

#### 49.2.6.10 HAL\_USART\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAPause (</b> <b>USART_HandleTypeDef * husart)</b>
Function Description	Pauses the DMA Transfer.

---

Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.6.11 HAL\_USART\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAResume (</b> <b>USART_HandleTypeDef * husart)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.6.12 HAL\_USART\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAStop (</b> <b>USART_HandleTypeDef * husart)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.6.13 HAL\_USART\_IRQHandler

Function Name	<b>void HAL_USART_IRQHandler ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	This function handles USART interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.6.14 HAL\_USART\_TxCpltCallback

Function Name	<b>void HAL_USART_TxCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 49.2.6.15 HAL\_USART\_TxHalfCpltCallback

Function Name	<b>void HAL_USART_TxHalfCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**49.2.6.16 HAL\_USART\_RxCpltCallback**

Function Name	<b>void HAL_USART_RxCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**49.2.6.17 HAL\_USART\_RxHalfCpltCallback**

Function Name	<b>void HAL_USART_RxHalfCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**49.2.6.18 HAL\_USART\_TxRxCpltCallback**

Function Name	<b>void HAL_USART_TxRxCpltCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Tx/Rx Transfers completed callback for the non-blocking process.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b> : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul>

---

Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 49.2.6.19 HAL\_USART\_ErrorCallback

Function Name	<b>void HAL_USART_ErrorCallback ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	<b>USART error callbacks.</b>
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : pointer to a <i>USART_HandleTypeDef</i> structure that contains the configuration information for the specified USART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None.</li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

#### 49.2.7 Peripheral State and Errors functions

##### 49.2.7.1 HAL\_USART\_GetState

Function Name	<b>HAL_USART_StateTypeDef HAL_USART_GetState ( <i>USART_HandleTypeDef</i> * <i>husart</i>)</b>
Function Description	Returns the USART state.
Parameters	<ul style="list-style-type: none"><li>• <b>husart</b> : pointer to a <i>USART_HandleTypeDef</i> structure that contains the configuration information for the specified USART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• <b>HAL state</b></li></ul>
Notes	<ul style="list-style-type: none"><li>• None.</li></ul>

##### 49.2.7.2 HAL\_USART\_GetError

Function Name	<code>uint32_t HAL_USART_GetError (<i>USART_HandleTypeDef</i>* <i>husart</i>)</code>
Function Description	Return the USART error code.
Parameters	<ul style="list-style-type: none"> <li>• <b><i>husart</i></b> : pointer to a <i>USART_HandleTypeDef</i> structure that contains the configuration information for the specified USART.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>USART Error Code</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 49.3 USART Firmware driver defines

### 49.3.1 USART

USART

***USART\_Clock***

- #define: ***USART\_CLOCK\_DISABLED*** ((*uint32\_t*)0x00000000)
- #define: ***USART\_CLOCK\_ENABLED*** ((*uint32\_t*)*USART\_CR2\_CLKEN*)

***USART\_Clock\_Phase***

- #define: ***USART\_PHASE\_1EDGE*** ((*uint32\_t*)0x00000000)
- #define: ***USART\_PHASE\_2EDGE*** ((*uint32\_t*)*USART\_CR2\_CPHA*)

***USART\_Clock\_Polarity***

- #define: ***USART\_POLARITY\_LOW*** ((*uint32\_t*)0x00000000)
- #define: ***USART\_POLARITY\_HIGH*** ((*uint32\_t*)*USART\_CR2\_CPOL*)

***USART\_Flags***

- #define: **USART\_FLAG\_TXE** ((*uint32\_t*)0x00000080)
- #define: **USART\_FLAG\_TC** ((*uint32\_t*)0x00000040)
- #define: **USART\_FLAG\_RXNE** ((*uint32\_t*)0x00000020)
- #define: **USART\_FLAG\_IDLE** ((*uint32\_t*)0x00000010)
- #define: **USART\_FLAG\_ORE** ((*uint32\_t*)0x00000008)
- #define: **USART\_FLAG\_NE** ((*uint32\_t*)0x00000004)
- #define: **USART\_FLAG\_FE** ((*uint32\_t*)0x00000002)
- #define: **USART\_FLAG\_PE** ((*uint32\_t*)0x00000001)

#### *USART Interrupt definition*

- #define: **USART\_IT\_PE** ((*uint32\_t*)0x10000100)
- #define: **USART\_IT\_TXE** ((*uint32\_t*)0x10000080)
- #define: **USART\_IT\_TC** ((*uint32\_t*)0x10000040)
- #define: **USART\_IT\_RXNE** ((*uint32\_t*)0x10000020)

- #define: **USART\_IT\_IDLE** ((*uint32\_t*)0x10000010)

- #define: **USART\_IT\_LBD** ((*uint32\_t*)0x20000040)

- #define: **USART\_IT\_CTS** ((*uint32\_t*)0x30000400)

- #define: **USART\_IT\_ERR** ((*uint32\_t*)0x30000001)

#### ***USART\_Last\_Bit***

- #define: **USART\_LASTBIT\_DISABLE** ((*uint32\_t*)0x00000000)

- #define: **USART\_LASTBIT\_ENABLE** ((*uint32\_t*)**USART\_CR2 LBCL**)

#### ***USART\_Mode***

- #define: **USART\_MODE\_RX** ((*uint32\_t*)**USART\_CR1 RE**)

- #define: **USART\_MODE\_TX** ((*uint32\_t*)**USART\_CR1 TE**)

- #define: **USART\_MODE\_TX\_RX** ((*uint32\_t*)(**USART\_CR1 TE** | **USART\_CR1 RE**))

#### ***USART\_NACK\_State***

- #define: **USARTNACK\_ENABLED** ((*uint32\_t*)**USART\_CR3 NACK**)

- #define: **USARTNACK\_DISABLED** ((*uint32\_t*)0x00000000)

***USART\_Parity***

- #define: ***USART\_PARITY\_NONE*** ((*uint32\_t*)0x00000000)
- #define: ***USART\_PARITY EVEN*** ((*uint32\_t*)***USART\_CR1\_PCE***)
- #define: ***USART\_PARITY ODD*** ((*uint32\_t*)(***USART\_CR1\_PCE | USART\_CR1\_PS***))

***USART\_Stop\_Bits***

- #define: ***USART\_STOPBITS\_1*** ((*uint32\_t*)0x00000000)
- #define: ***USART\_STOPBITS\_0\_5*** ((*uint32\_t*)***USART\_CR2\_STOP\_0***)
- #define: ***USART\_STOPBITS\_2*** ((*uint32\_t*)***USART\_CR2\_STOP\_1***)
- #define: ***USART\_STOPBITS\_1\_5*** ((*uint32\_t*)(***USART\_CR2\_STOP\_0 | USART\_CR2\_STOP\_1***))

***USART\_Word\_Length***

- #define: ***USART\_WORDLENGTH\_8B*** ((*uint32\_t*)0x00000000)
- #define: ***USART\_WORDLENGTH\_9B*** ((*uint32\_t*)***USART\_CR1\_M***)

## 50 HAL WWDG Generic Driver

### 50.1 WWDG Firmware driver registers structures

#### 50.1.1 WWDG\_HandleTypeDefDef

*WWDG\_HandleTypeDefDef* is defined in the `stm32f4xx_hal_wwdg.h`

##### Data Fields

- *WWDG\_TypeDef \* Instance*
- *WWDG\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_WWDG\_StateTypeDef State*

##### Field Documentation

- *WWDG\_TypeDef\* WWDG\_HandleTypeDefDef::Instance*
  - Register base address
- *WWDG\_InitTypeDef WWDG\_HandleTypeDefDef::Init*
  - WWDG required parameters
- *HAL\_LockTypeDef WWDG\_HandleTypeDefDef::Lock*
  - WWDG locking object
- *\_\_IO HAL\_WWDG\_StateTypeDef WWDG\_HandleTypeDefDef::State*
  - WWDG communication state

#### 50.1.2 WWDG\_InitTypeDef

*WWDG\_InitTypeDef* is defined in the `stm32f4xx_hal_wwdg.h`

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Window*
- *uint32\_t Counter*

##### Field Documentation

- *uint32\_t WWDG\_InitTypeDef::Prescaler*
  - Specifies the prescaler. 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

### 50.1.3 WWDG\_TypeDef

*WWDG\_TypeDef* is defined in the stm32f439xx.h

#### Data Fields

- `__IO uint32_t CR`
- `__IO uint32_t CFR`
- `__IO uint32_t SR`

#### Field Documentation

- `__IO uint32_t WWDG_TypeDef::CR`
  - WWDG Control register, Address offset: 0x00
- `__IO uint32_t WWDG_TypeDef::CFR`
  - WWDG Configuration register, Address offset: 0x04
- `__IO uint32_t WWDG_TypeDef::SR`
  - WWDG Status register, Address offset: 0x08

## 50.2 WWDG Firmware driver API description

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

### 50.2.1 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_DelInit()`
- `HAL_WWDG_MspInit()`
- `HAL_WWDG_MspDelInit()`

### 50.2.2 IO operation functions

This section provides functions allowing to:

- Start the WWDG.
- Refresh the WWDG.
- handle WWDG interrupt request.
- `HAL_WWDG_Start()`
- `HAL_WWDG_Start_IT()`
- `HAL_WWDG_Refresh()`

- [\*HAL\\_WWDG\\_IRQHandler\(\)\*](#)
- [\*HAL\\_WWDG\\_WakeupCallback\(\)\*](#)

### 50.2.3 Peripheral State functions

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

- [\*HAL\\_WWDG\\_GetState\(\)\*](#)

### 50.2.4 Initialization and de-initialization functions

#### 50.2.4.1 HAL\_WWDG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Init (</b> <a href="#"><i>WWDG_HandleTypeDef * hwdg</i></a> <b>)</b>
Function Description	Initializes the WWDG according to the specified parameters in the <i>WWDG_InitTypeDef</i> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg</b> : pointer to a <i>WWDG_HandleTypeDef</i> structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 50.2.4.2 HAL\_WWDG\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_DeInit (</b> <a href="#"><i>WWDG_HandleTypeDef * hwdg</i></a> <b>)</b>
Function Description	Deinitializes the WWDG peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg</b> : pointer to a <i>WWDG_HandleTypeDef</i> structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 50.2.4.3 HAL\_WWDG\_MspInit

Function Name	<b>void HAL_WWDG_MspInit ( <i>WWDG_HandleTypeDef</i> * hwdg)</b>
Function Description	Initializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

### 50.2.4.4 HAL\_WWDG\_MspDeInit

Function Name	<b>void HAL_WWDG_MspDeInit ( <i>WWDG_HandleTypeDef</i> * hwdg)</b>
Function Description	Deinitializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

## 50.2.5 IO operation functions

### 50.2.5.1 HAL\_WWDG\_Start

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Start ( <i>WWDG_HandleTypeDef</i> * hwdg)</b>
Function Description	Starts the WWDG.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>

## Notes

- None.

**50.2.5.2 HAL\_WWDG\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Start_IT (</b> <b>WWDG_HandleTypeDef * hwdg)</b>
Function Description	Starts the WWDG with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**50.2.5.3 HAL\_WWDG\_Refresh**

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Refresh (</b> <b>WWDG_HandleTypeDef * hwdg, uint32_t Counter)</b>
Function Description	Refreshes the WWDG.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• <b>HAL status</b></li> </ul>
Notes	<ul style="list-style-type: none"> <li>• None.</li> </ul>

**50.2.5.4 HAL\_WWDG\_IRQHandler**

Function Name	<b>void HAL_WWDG_IRQHandler (</b> <b>WWDG_HandleTypeDef * hwdg)</b>
Function Description	Handles WWDG interrupt request.

Parameters	<ul style="list-style-type: none"> <li><b>hwdwg</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>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 <code>__HAL_WWDG_ENABLE_IT()</code> 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.</li> </ul>

### 50.2.5.5 HAL\_WWDG\_WakeupCallback

Function Name	<code>void HAL_WWDG_WakeupCallback ( <i>WWDG_HandleTypeDef</i> * hwdwg)</code>
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> <li><b>hwdwg</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None.</li> </ul>
Notes	<ul style="list-style-type: none"> <li>None.</li> </ul>

## 50.2.6 Peripheral State functions

### 50.2.6.1 HAL\_WWDG\_GetState

Function Name	<code>HAL_WWDG_StateTypeDef HAL_WWDG_GetState ( <i>WWDG_HandleTypeDef</i> * hwdwg)</code>
Function Description	Returns the WWDG state.
Parameters	<ul style="list-style-type: none"> <li><b>hwdwg</b> : pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li><b>HAL state</b></li> </ul>

Notes

- None.

## 50.3 WWDG Firmware driver defines

### 50.3.1 WWDG

WWDG

**WWDG\_BitAddress\_AliasRegion**

- #define: **CFR\_BASE** (*uint32\_t*)**(WWDG\_BASE + 0x04)**

**WWDG\_Flag\_definition**

- #define: **WWDG\_FLAG\_EWIF** ((*uint32\_t*)0x0001)

*Early wakeup interrupt flag*

**WWDG\_Interrupt\_definition**

- #define: **WWDG\_IT\_EWI** ((*uint32\_t*)**WWDG\_CFR\_EWI**)

**WWDG\_Prescaler**

- #define: **WWDG\_PRESCALER\_1** ((*uint32\_t*)0x00000000)

*WWDG counter clock = (PCLK1/4096)/1*

- #define: **WWDG\_PRESCALER\_2** ((*uint32\_t*)0x00000080)

*WWDG counter clock = (PCLK1/4096)/2*

- #define: **WWDG\_PRESCALER\_4** ((*uint32\_t*)0x00000100)

*WWDG counter clock = (PCLK1/4096)/4*

- #define: **WWDG\_PRESCALER\_8** ((*uint32\_t*)0x00000180)

*WWDG counter clock = (PCLK1/4096)/8*

## 51 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 require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, ...). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL\_UART\_Init() then HAL\_UART\_Transmit() or HAL\_UART\_Receive().

#### Which STM32F4 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32F4 devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to [Section 4.4: "Devices supported by HAL drivers"](#).

#### What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

### Architecture

#### How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32f4xx\_hal\_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration, Ethernet parameters configuration...)

A template is provided in the HAL drivers folders (stm32f4xx\_hal\_conf\_template.c).

### Which header files should I include in my application to use the HAL drivers?

Only stm32f4xx\_hal.h file has to be included.

### What is the difference between stm32f4xx\_hal\_ppp.c/h and stm32f4xx\_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 (stm32f4xx\_hal\_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32f4xx\_hal\_ppp\_ex.c): It includes the specific APIs for specific device part number or family.

### Is it possible to use the APIs available in stm32f4xx\_ll\_ppp.c?

These APIs cannot be used directly because they are internal and offer services to upper layer drivers. As an example stm32f4xx\_ll\_fmc.c/h driver is used by stm32f4xx\_hal\_sram.c, stm32f4xx\_hal\_nor.c, stm32f4xx\_hal\_nand.c and stm32f4xx\_hal\_sdram.c drivers.

### 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\_stm32f4xx.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 stm32f4xx\_hal\_msp.c. A template is provided in the HAL driver folders (stm32f4xx\_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 `stm32f4xx_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 stm32f4xx\_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 stm32f4xx\_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.

## 52 Revision history

**Table 16: Document revision history**

Date	Revision	Changes
09-May-2014	1	Initial release.

**Please Read Carefully**

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

**UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.**

**ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER'S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR "AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL" INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.**

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2014 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan -  
Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

[www.st.com](http://www.st.com)