

## Description of STM32F2xx HAL drivers

### Introduction

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

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF2 for stm32f2 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>41</b>
<b>2</b>	<b>Overview of HAL drivers .....</b>	<b>43</b>
2.1	HAL and user-application files.....	43
2.1.1	HAL driver files .....	43
2.1.2	User-application files .....	45
2.2	HAL data structures .....	46
2.2.1	Peripheral handle structures .....	46
2.2.2	Initialization and configuration structure .....	48
2.2.3	Specific process structures .....	48
2.3	API classification .....	48
2.4	Devices supported by HAL drivers .....	49
2.5	HAL drivers rules.....	50
2.5.1	HAL API naming rules .....	50
2.5.2	HAL general naming rules .....	51
2.5.3	HAL interrupt handler and callback functions.....	53
2.6	HAL generic APIs.....	53
2.7	HAL extension APIs .....	55
2.7.1	HAL extension model overview .....	55
2.7.2	HAL extension model cases .....	55
2.8	File inclusion model.....	57
2.9	HAL common resources.....	57
2.10	HAL configuration.....	58
2.11	HAL system peripheral handling .....	60
2.11.1	Clock.....	60
2.11.2	GPIOs.....	60
2.11.3	Cortex NVIC and SysTick timer.....	62
2.11.4	PWR .....	62
2.11.5	EXTI.....	62
2.11.6	DMA.....	63
2.12	How to use HAL drivers .....	65
2.12.1	HAL usage models .....	65
2.12.2	HAL initialization .....	66
2.12.3	HAL IO operation process .....	68
2.12.4	Timeout and error management.....	71
<b>3</b>	<b>HAL System Driver .....</b>	<b>75</b>

3.1	HAL Firmware driver API description .....	75
3.1.1	How to use this driver .....	75
3.1.2	Initialization and de-initialization functions .....	75
3.1.3	HAL Control functions.....	75
3.1.4	HAL_Init.....	76
3.1.5	HAL_DelInit .....	76
3.1.6	HAL_MspInit .....	76
3.1.7	HAL_MspDeInit .....	76
3.1.8	HAL_InitTick .....	77
3.1.9	HAL_IncTick .....	77
3.1.10	HAL_GetTick .....	77
3.1.11	HAL_Delay .....	77
3.1.12	HAL_SuspendTick.....	78
3.1.13	HAL_ResumeTick.....	78
3.1.14	HAL_GetHalVersion .....	78
3.1.15	HAL_GetREVID .....	78
3.1.16	HAL_GetDEVID.....	78
3.1.17	HAL_DBGMCU_EnableDBGSleepMode .....	79
3.1.18	HAL_DBGMCU_DisableDBGSleepMode .....	79
3.1.19	HAL_DBGMCU_EnableDBGStopMode .....	79
3.1.20	HAL_DBGMCU_DisableDBGStopMode .....	79
3.1.21	HAL_DBGMCU_EnableDBGStandbyMode .....	79
3.1.22	HAL_DBGMCU_DisableDBGStandbyMode .....	79
3.1.23	HAL_EnableCompensationCell.....	79
3.1.24	HAL_DisableCompensationCell .....	80
3.2	HAL Firmware driver defines.....	80
3.2.1	HAL.....	80
4	HAL ADC Generic Driver.....	83
4.1	ADC Firmware driver registers structures .....	83
4.1.1	ADC_InitTypeDef.....	83
4.1.2	ADC_ChannelConfTypeDef .....	84
4.1.3	ADC_AnalogWDGConfTypeDef.....	85
4.1.4	ADC_HandleTypeDef.....	86
4.2	ADC Firmware driver API description.....	86
4.2.1	ADC Peripheral features.....	86
4.2.2	How to use this driver .....	87
4.2.3	Initialization and de-initialization functions .....	89
4.2.4	IO operation functions .....	89

4.2.5	Peripheral Control functions .....	90
4.2.6	Peripheral State and errors functions.....	90
4.2.7	HAL_ADC_Init .....	90
4.2.8	HAL_ADC_DelInit.....	90
4.2.9	HAL_ADC_MspInit .....	91
4.2.10	HAL_ADC_MspDeInit.....	91
4.2.11	HAL_ADC_Start .....	91
4.2.12	HAL_ADC_Stop.....	91
4.2.13	HAL_ADC_PollForConversion .....	91
4.2.14	HAL_ADC_PollForEvent .....	92
4.2.15	HAL_ADC_Start_IT .....	92
4.2.16	HAL_ADC_Stop_IT .....	92
4.2.17	HAL_ADC_IRQHandler.....	93
4.2.18	HAL_ADC_Start_DMA .....	93
4.2.19	HAL_ADC_Stop_DMA.....	93
4.2.20	HAL_ADC_GetValue .....	93
4.2.21	HAL_ADC_ConvCpltCallback .....	93
4.2.22	HAL_ADC_ConvHalfCpltCallback .....	94
4.2.23	HAL_ADC_LevelOutOfWindowCallback .....	94
4.2.24	HAL_ADC_ErrorCallback .....	94
4.2.25	HAL_ADC_ConfigChannel .....	94
4.2.26	HAL_ADC_AnalogWDGConfig .....	95
4.2.27	HAL_ADC_GetState.....	95
4.2.28	HAL_ADC_GetError .....	95
4.3	ADC Firmware driver defines .....	95
4.3.1	ADC .....	95
<b>5</b>	<b>HAL ADC Extension Driver .....</b>	<b>106</b>
5.1	ADCEx Firmware driver registers structures .....	106
5.1.1	ADC_InjectionConfTypeDef .....	106
5.1.2	ADC_MultiModeTypeDef.....	107
5.2	ADCEx Firmware driver API description .....	108
5.2.1	How to use this driver .....	108
5.2.2	Extended features functions.....	109
5.2.3	HAL_ADCEx_InjectedStart .....	110
5.2.4	HAL_ADCEx_InjectedStart_IT .....	110
5.2.5	HAL_ADCEx_InjectedStop.....	110
5.2.6	HAL_ADCEx_InjectedPollForConversion .....	110
5.2.7	HAL_ADCEx_InjectedStop_IT .....	111

5.2.8	HAL_ADCEx_InjectedGetValue .....	111
5.2.9	HAL_ADCEx_MultiModeStart_DMA .....	111
5.2.10	HAL_ADCEx_MultiModeStop_DMA.....	112
5.2.11	HAL_ADCEx_MultiModeGetValue .....	112
5.2.12	HAL_ADCEx_InjectedConvCpltCallback .....	112
5.2.13	HAL_ADCEx_InjectedConfigChannel .....	112
5.2.14	HAL_ADCEx_MultiModeConfigChannel .....	113
5.3	ADCEx Firmware driver defines .....	113
5.3.1	ADCEx .....	113
<b>6</b>	<b>HAL CAN Generic Driver.....</b>	<b>116</b>
6.1	CAN Firmware driver registers structures .....	116
6.1.1	CAN_InitTypeDef.....	116
6.1.2	CAN_FilterConfTypeDef.....	117
6.1.3	CanTxMsgTypeDef.....	118
6.1.4	CanRxMsgTypeDef .....	118
6.1.5	CAN_HandleTypeDef.....	119
6.2	CAN Firmware driver API description.....	120
6.2.1	How to use this driver .....	120
6.2.2	Initialization and de-initialization functions .....	121
6.2.3	IO operation functions .....	121
6.2.4	Peripheral State and Error functions .....	121
6.2.5	HAL_CAN_Init .....	122
6.2.6	HAL_CAN_ConfigFilter.....	122
6.2.7	HAL_CAN_DeInit.....	122
6.2.8	HAL_CAN_MspInit .....	122
6.2.9	HAL_CAN_MspDeInit.....	122
6.2.10	HAL_CAN_Transmit.....	123
6.2.11	HAL_CAN_Transmit_IT.....	123
6.2.12	HAL_CAN_Receive .....	123
6.2.13	HAL_CAN_Receive_IT.....	123
6.2.14	HAL_CAN_Sleep.....	123
6.2.15	HAL_CAN_WakeUp .....	124
6.2.16	HAL_CAN_IRQHandler.....	124
6.2.17	HAL_CAN_TxCpltCallback.....	124
6.2.18	HAL_CAN_RxCpltCallback .....	124
6.2.19	HAL_CAN_ErrorCallback .....	124
6.2.20	HAL_CAN_GetState.....	125
6.2.21	HAL_CAN_GetError .....	125

6.3	CAN Firmware driver defines .....	125
6.3.1	CAN .....	125
<b>7</b>	<b>HAL CORTEX Generic Driver.....</b>	<b>133</b>
7.1	CORTEX Firmware driver registers structures .....	133
7.1.1	MPU_Region_InitTypeDef.....	133
7.2	CORTEX Firmware driver API description .....	134
7.2.1	How to use this driver .....	134
7.2.2	Initialization and de-initialization functions .....	135
7.2.3	Peripheral Control functions .....	135
7.2.4	HAL_NVIC_SetPriorityGrouping .....	135
7.2.5	HAL_NVIC_SetPriority .....	135
7.2.6	HAL_NVIC_EnableIRQ .....	136
7.2.7	HAL_NVIC_DisableIRQ.....	136
7.2.8	HAL_NVIC_SystemReset.....	136
7.2.9	HAL_SYSTICK_Config.....	136
7.2.10	HAL_MPUMemoryRegionConfig.....	137
7.2.11	HAL_NVIC_GetPriorityGrouping .....	137
7.2.12	HAL_NVIC_GetPriority .....	137
7.2.13	HAL_NVIC_SetPendingIRQ .....	137
7.2.14	HAL_NVIC_GetPendingIRQ .....	138
7.2.15	HAL_NVIC_ClearPendingIRQ.....	138
7.2.16	HAL_NVIC_GetActive .....	138
7.2.17	HAL_SYSTICK_CLKSourceConfig .....	138
7.2.18	HAL_SYSTICK_IRQHandler .....	139
7.2.19	HAL_SYSTICK_Callback .....	139
7.3	CORTEX Firmware driver defines .....	139
7.3.1	CORTEX.....	139
<b>8</b>	<b>HAL CRC Generic Driver.....</b>	<b>143</b>
8.1	CRC Firmware driver registers structures .....	143
8.1.1	CRC_HandleTypeDef.....	143
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 .....	144
8.2.4	Peripheral State functions .....	144
8.2.5	HAL_CRC_Init .....	144
8.2.6	HAL_CRC_DeInit .....	144

8.2.7	HAL_CRC_MspInit .....	144
8.2.8	HAL_CRC_MspDelInit.....	144
8.2.9	HAL_CRC_Accumulate.....	145
8.2.10	HAL_CRC_Calculate.....	145
8.2.11	HAL_CRC_GetState.....	145
8.3	CRC Firmware driver defines .....	145
8.3.1	CRC .....	145
<b>9</b>	<b>HAL CRYP Generic Driver.....</b>	<b>147</b>
9.1	CRYP Firmware driver registers structures .....	147
9.1.1	CRYP_InitTypeDef .....	147
9.1.2	CRYP_HandleTypeDef.....	147
9.2	CRYP Firmware driver API description .....	148
9.2.1	How to use this driver .....	148
9.2.2	Initialization and de-initialization functions .....	149
9.2.3	AES processing functions .....	149
9.2.4	DES processing functions .....	150
9.2.5	TDES processing functions .....	150
9.2.6	DMA callback functions .....	151
9.2.7	CRYP IRQ handler management.....	151
9.2.8	Peripheral State functions .....	151
9.2.9	HAL_CRYP_Init.....	151
9.2.10	HAL_CRYP_DelInit .....	152
9.2.11	HAL_CRYP_MspInit .....	152
9.2.12	HAL_CRYP_MspDelInit .....	152
9.2.13	HAL_CRYP_AESECB_Encrypt.....	152
9.2.14	HAL_CRYP_AESCBC_Encrypt .....	152
9.2.15	HAL_CRYP_AESCTR_Encrypt.....	153
9.2.16	HAL_CRYP_AESECB_Decrypt .....	153
9.2.17	HAL_CRYP_AESCBC_Decrypt .....	153
9.2.18	HAL_CRYP_AESCTR_Decrypt .....	154
9.2.19	HAL_CRYP_AESECB_Encrypt_IT .....	154
9.2.20	HAL_CRYP_AESCBC_Encrypt_IT .....	154
9.2.21	HAL_CRYP_AESCTR_Encrypt_IT .....	155
9.2.22	HAL_CRYP_AESECB_Decrypt_IT .....	155
9.2.23	HAL_CRYP_AESCBC_Decrypt_IT .....	155
9.2.24	HAL_CRYP_AESCTR_Decrypt_IT .....	156
9.2.25	HAL_CRYP_AESECB_Encrypt_DMA.....	156
9.2.26	HAL_CRYP_AESCBC_Encrypt_DMA .....	156

9.2.27	HAL_CRYP_AESCTR_Encrypt_DMA.....	156
9.2.28	HAL_CRYP_AESECB_Decrypt_DMA .....	157
9.2.29	HAL_CRYP_AESCBC_Decrypt_DMA .....	157
9.2.30	HAL_CRYP_AESCTR_Decrypt_DMA .....	157
9.2.31	HAL_CRYP_DESECB_Encrypt .....	158
9.2.32	HAL_CRYP_DESECB_Decrypt .....	158
9.2.33	HAL_CRYP_DESCBC_Encrypt .....	158
9.2.34	HAL_CRYP_DESCBC_Decrypt .....	158
9.2.35	HAL_CRYP_DESECB_Encrypt_IT .....	159
9.2.36	HAL_CRYP_DESCBC_Encrypt_IT .....	159
9.2.37	HAL_CRYP_DESECB_Decrypt_IT .....	159
9.2.38	HAL_CRYP_DESCBC_Decrypt_IT .....	160
9.2.39	HAL_CRYP_DESECB_Encrypt_DMA .....	160
9.2.40	HAL_CRYP_DESCBC_Encrypt_DMA .....	160
9.2.41	HAL_CRYP_DESECB_Decrypt_DMA .....	160
9.2.42	HAL_CRYP_DESCBC_Decrypt_DMA .....	161
9.2.43	HAL_CRYP_TDESECB_Encrypt .....	161
9.2.44	HAL_CRYP_TDESECB_Decrypt .....	161
9.2.45	HAL_CRYP_TDESCBC_Encrypt .....	162
9.2.46	HAL_CRYP_TDESCBC_Decrypt .....	162
9.2.47	HAL_CRYP_TDESECB_Encrypt_IT .....	162
9.2.48	HAL_CRYP_TDESCBC_Encrypt_IT .....	163
9.2.49	HAL_CRYP_TDESECB_Decrypt_IT .....	163
9.2.50	HAL_CRYP_TDESCBC_Decrypt_IT .....	163
9.2.51	HAL_CRYP_TDESECB_Encrypt_DMA .....	163
9.2.52	HAL_CRYP_TDESCBC_Encrypt_DMA .....	164
9.2.53	HAL_CRYP_TDESECB_Decrypt_DMA .....	164
9.2.54	HAL_CRYP_TDESCBC_Decrypt_DMA .....	164
9.2.55	HAL_CRYP_InCpltCallback .....	165
9.2.56	HAL_CRYP_OutCpltCallback .....	165
9.2.57	HAL_CRYP_ErrorCallback .....	165
9.2.58	HAL_CRYP_IRQHandler .....	165
9.2.59	HAL_CRYP_GetState .....	165
9.3	CRYP Firmware driver defines.....	166
9.3.1	CRYP .....	166
<b>10</b>	<b>HAL DAC Generic Driver.....</b>	<b>170</b>
10.1	DAC Firmware driver registers structures .....	170
10.1.1	DAC_HandleTypeDef .....	170

10.1.2	DAC_ChannelConfTypeDef .....	170
10.2	DAC Firmware driver API description.....	171
10.2.1	DAC Peripheral features.....	171
10.2.2	How to use this driver.....	172
10.2.3	Initialization and de-initialization functions .....	173
10.2.4	IO operation functions .....	173
10.2.5	Peripheral Control functions .....	173
10.2.6	Peripheral State and Errors functions .....	173
10.2.7	HAL_DAC_Init .....	174
10.2.8	HAL_DAC_DelInit.....	174
10.2.9	HAL_DAC_MspInit .....	174
10.2.10	HAL_DAC_MspDeInit.....	174
10.2.11	HAL_DAC_Start .....	174
10.2.12	HAL_DAC_Stop.....	175
10.2.13	HAL_DAC_Start_DMA .....	175
10.2.14	HAL_DAC_Stop_DMA.....	175
10.2.15	HAL_DAC_GetValue .....	176
10.2.16	HAL_DAC_IRQHandler.....	176
10.2.17	HAL_DAC_ConvCpltCallbackCh1 .....	176
10.2.18	HAL_DAC_ConvHalfCpltCallbackCh1 .....	176
10.2.19	HAL_DAC_ErrorCallbackCh1 .....	177
10.2.20	HAL_DAC_DMAUnderrunCallbackCh1 .....	177
10.2.21	HAL_DAC_ConfigChannel .....	177
10.2.22	HAL_DAC_SetValue .....	177
10.2.23	HAL_DAC_GetState.....	178
10.2.24	HAL_DAC_GetError .....	178
10.2.25	HAL_DAC_IRQHandler.....	178
10.2.26	HAL_DAC_ConvCpltCallbackCh1 .....	178
10.2.27	HAL_DAC_ConvHalfCpltCallbackCh1 .....	178
10.2.28	HAL_DAC_ErrorCallbackCh1 .....	179
10.2.29	HAL_DAC_DMAUnderrunCallbackCh1 .....	179
10.3	DAC Firmware driver defines .....	179
10.3.1	DAC .....	179
11	HAL DAC Extension Driver .....	184
11.1	DACEEx Firmware driver API description .....	184
11.1.1	How to use this driver.....	184
11.1.2	Extended features functions .....	184
11.1.3	HAL_DACEEx_DualGetValue .....	184

11.1.4	HAL_DACEx_TriangleWaveGenerate .....	184
11.1.5	HAL_DACEx_NoiseWaveGenerate .....	185
11.1.6	HAL_DACEx_DualSetValue.....	186
11.1.7	HAL_DACEx_ConvCpltCallbackCh2 .....	186
11.1.8	HAL_DACEx_ConvHalfCpltCallbackCh2 .....	186
11.1.9	HAL_DACEx_ErrorCallbackCh2 .....	187
11.1.10	HAL_DACEx_DMAUnderrunCallbackCh2 .....	187
11.2	DACEx Firmware driver defines .....	187
11.2.1	DACEx .....	187
<b>12</b>	<b>HAL DCMI Generic Driver .....</b>	<b>189</b>
12.1	DCMI Firmware driver registers structures.....	189
12.1.1	DCMI_CodesInitTypeDef.....	189
12.1.2	DCMI_InitTypeDef .....	189
12.1.3	DCMI_HandleTypeDef .....	190
12.2	DCMI Firmware driver API description .....	191
12.2.1	How to use this driver .....	191
12.2.2	Initialization and Configuration functions.....	191
12.2.3	IO operation functions .....	192
12.2.4	Peripheral Control functions .....	192
12.2.5	Peripheral State and Errors functions .....	192
12.2.6	HAL_DCMI_Init.....	192
12.2.7	HAL_DCMI_DeInit .....	193
12.2.8	HAL_DCMI_MspInit.....	193
12.2.9	HAL_DCMI_MspDeInit .....	193
12.2.10	HAL_DCMI_Start_DMA.....	193
12.2.11	HAL_DCMI_Stop .....	193
12.2.12	HAL_DCMI_IRQHandler .....	194
12.2.13	HAL_DCMI_ErrorCallback .....	194
12.2.14	HAL_DCMI_LineEventCallback .....	194
12.2.15	HAL_DCMI_VsyncEventCallback .....	194
12.2.16	HAL_DCMI_FrameEventCallback .....	194
12.2.17	HAL_DCMI_ConfigCROP.....	195
12.2.18	HAL_DCMI_DisableCROP .....	195
12.2.19	HAL_DCMI_EnableCROP .....	195
12.2.20	HAL_DCMI_GetState .....	195
12.2.21	HAL_DCMI_GetError.....	195
12.3	DCMI Firmware driver defines.....	196
12.3.1	DCMI.....	196

<b>13 HAL DMA Generic Driver .....</b>	<b>201</b>
13.1 DMA Firmware driver registers structures .....	201
13.1.1 DMA_InitTypeDef .....	201
13.1.2 __DMA_HandleTypeDef.....	202
13.2 DMA Firmware driver API description .....	203
13.2.1 How to use this driver .....	203
13.2.2 Initialization and de-initialization functions .....	204
13.2.3 IO operation functions .....	204
13.2.4 State and Errors functions .....	205
13.2.5 HAL_DMA_Init.....	205
13.2.6 HAL_DMA_DeInit .....	205
13.2.7 HAL_DMA_Start .....	205
13.2.8 HAL_DMA_Start_IT.....	206
13.2.9 HAL_DMA_Abort .....	206
13.2.10 HAL_DMA_PollForTransfer.....	206
13.2.11 HAL_DMA_IRQHandler.....	206
13.2.12 HAL_DMA_GetState .....	207
13.2.13 HAL_DMA_GetError .....	207
13.3 DMA Firmware driver defines.....	207
13.3.1 DMA.....	207
<b>14 HAL DMA Extension Driver.....</b>	<b>211</b>
14.1 DMAEx Firmware driver API description .....	211
14.1.1 How to use this driver.....	211
14.1.2 Extended features functions .....	211
14.1.3 HAL_DMAEx_MultiBufferStart .....	211
14.1.4 HAL_DMAEx_MultiBufferStart_IT .....	211
14.1.5 HAL_DMAEx_ChangeMemory.....	212
<b>15 HAL ETH Generic Driver .....</b>	<b>213</b>
15.1 ETH Firmware driver registers structures.....	213
15.1.1 ETH_InitTypeDef .....	213
15.1.2 ETH_MACInitTypeDef .....	213
15.1.3 ETH_DMAMainInitTypeDef .....	216
15.1.4 ETH_DMADescTypeDef .....	217
15.1.5 ETH_DMARxFrameInfos .....	218
15.1.6 ETH_HandleTypeDef .....	219
15.2 ETH Firmware driver API description .....	219
15.2.1 How to use this driver .....	219

15.2.2	Initialization and de-initialization functions .....	220
15.2.3	IO operation functions .....	220
15.2.4	Peripheral Control functions .....	221
15.2.5	Peripheral State functions .....	221
15.2.6	HAL_ETH_Init.....	221
15.2.7	HAL_ETH_DelInit.....	221
15.2.8	HAL_ETH_DMATxDescListInit.....	221
15.2.9	HAL_ETH_DMARxDescListInit .....	222
15.2.10	HAL_ETH_MspInit.....	222
15.2.11	HAL_ETH_MspDeInit .....	222
15.2.12	HAL_ETH_TransmitFrame .....	222
15.2.13	HAL_ETH_GetReceivedFrame .....	222
15.2.14	HAL_ETH_GetReceivedFrame_IT .....	223
15.2.15	HAL_ETH_IRQHandler .....	223
15.2.16	HAL_ETH_TxCpltCallback .....	223
15.2.17	HAL_ETH_RxCpltCallback.....	223
15.2.18	HAL_ETH_ErrorCallback.....	223
15.2.19	HAL_ETH_ReadPHYRegister .....	224
15.2.20	HAL_ETH_WritePHYRegister .....	224
15.2.21	HAL_ETH_Start.....	224
15.2.22	HAL_ETH_Stop .....	224
15.2.23	HAL_ETH_ConfigMAC .....	225
15.2.24	HAL_ETH_ConfigDMA .....	225
15.2.25	HAL_ETH_GetState .....	225
15.3	ETH Firmware driver defines.....	225
15.3.1	ETH.....	225
<b>16</b>	<b>HAL FLASH Generic Driver.....</b>	<b>257</b>
16.1	FLASH Firmware driver registers structures .....	257
16.1.1	FLASH_ProcessTypeDef .....	257
16.2	FLASH Firmware driver API description.....	257
16.2.1	FLASH peripheral features .....	257
16.2.2	How to use this driver .....	258
16.2.3	Programming operation functions .....	258
16.2.4	Peripheral Control functions .....	258
16.2.5	Peripheral Errors functions .....	258
16.2.6	HAL_FLASH_Program .....	259
16.2.7	HAL_FLASH_Program_IT .....	259
16.2.8	HAL_FLASH_IRQHandler .....	259

16.2.9	HAL_FLASH_EndOfOperationCallback .....	259
16.2.10	HAL_FLASH_OperationErrorHandler .....	259
16.2.11	HAL_FLASH_Unlock .....	260
16.2.12	HAL_FLASH_Lock .....	260
16.2.13	HAL_FLASH_OB_Unlock .....	260
16.2.14	HAL_FLASH_OB_Lock .....	260
16.2.15	HAL_FLASH_OB_Launch .....	260
16.2.16	HAL_FLASH_GetError .....	260
16.2.17	FLASH_WaitForLastOperation .....	261
16.3	FLASH Firmware driver defines .....	261
16.3.1	FLASH .....	261
<b>17</b>	<b>HAL FLASH Extension Driver .....</b>	<b>267</b>
17.1	FLASHEx Firmware driver registers structures .....	267
17.1.1	FLASH_EraseInitTypeDef .....	267
17.1.2	FLASH_OBProgramInitTypeDef .....	267
17.2	FLASHEx Firmware driver API description .....	268
17.2.1	Flash Extension features .....	268
17.2.2	How to use this driver .....	268
17.2.3	Extended programming operation functions .....	268
17.2.4	HAL_FLASHEx_Erase .....	268
17.2.5	HAL_FLASHEx_Erase_IT .....	269
17.2.6	HAL_FLASHEx_OBProgram .....	269
17.2.7	HAL_FLASHEx_OBGetConfig .....	269
17.3	FLASHEx Firmware driver defines .....	269
17.3.1	FLASHEx .....	269
<b>18</b>	<b>HAL GPIO Generic Driver .....</b>	<b>273</b>
18.1	GPIO Firmware driver registers structures .....	273
18.1.1	GPIO_InitTypeDef .....	273
18.2	GPIO Firmware driver API description .....	273
18.2.1	GPIO Peripheral features .....	273
18.2.2	How to use this driver .....	274
18.2.3	Initialization and de-initialization functions .....	274
18.2.4	IO operation functions .....	275
18.2.5	HAL_GPIO_Init .....	275
18.2.6	HAL_GPIO_DelInit .....	275
18.2.7	HAL_GPIO_ReadPin .....	275
18.2.8	HAL_GPIO_WritePin .....	275

18.2.9	HAL_GPIO_TogglePin .....	276
18.2.10	HAL_GPIO_LockPin.....	276
18.2.11	HAL_GPIO_EXTI_IRQHandler .....	276
18.2.12	HAL_GPIO_EXTI_Callback.....	276
18.3	GPIO Firmware driver defines.....	277
18.3.1	GPIO .....	277
<b>19</b>	<b>HAL GPIO Extension Driver .....</b>	<b>282</b>
19.1	GPIOEx Firmware driver defines.....	282
19.1.1	GPIOEx .....	282
<b>20</b>	<b>HAL HASH Generic Driver .....</b>	<b>283</b>
20.1	HASH Firmware driver registers structures .....	283
20.1.1	HASH_InitTypeDef .....	283
20.1.2	HASH_HandleTypeDef.....	283
20.2	HASH Firmware driver API description .....	284
20.2.1	How to use this driver .....	284
20.2.2	HASH processing using polling mode functions .....	285
20.2.3	HASH processing using interrupt mode functions.....	285
20.2.4	HASH processing using DMA mode functions .....	285
20.2.5	HMAC processing using polling mode functions .....	286
20.2.6	HMAC processing using DMA mode functions .....	286
20.2.7	Peripheral State functions .....	286
20.2.8	Initialization and de-initialization functions .....	286
20.2.9	HAL_HASH_MD5_Start .....	287
20.2.10	HAL_HASH_MD5_Accumulate .....	287
20.2.11	HAL_HASH_SHA1_Start.....	287
20.2.12	HAL_HASH_SHA1_Accumulate .....	288
20.2.13	HAL_HASH_MD5_Start_IT .....	288
20.2.14	HAL_HASH_SHA1_Start_IT .....	288
20.2.15	HAL_HASH_IRQHandler.....	289
20.2.16	HAL_HMAC_SHA1_Start.....	289
20.2.17	HAL_HMAC_MD5_Start.....	289
20.2.18	HAL_HASH_MD5_Start_DMA .....	289
20.2.19	HAL_HASH_MD5_Finish .....	290
20.2.20	HAL_HASH_SHA1_Start_DMA .....	290
20.2.21	HAL_HASH_SHA1_Finish .....	290
20.2.22	HAL_HASH_SHA1_Start_IT .....	291
20.2.23	HAL_HASH_MD5_Start_IT .....	291
20.2.24	HAL_HMAC_MD5_Start.....	291

20.2.25	HAL_HMAC_SHA1_Start .....	292
20.2.26	HAL_HASH_SHA1_Start_DMA .....	292
20.2.27	HAL_HASH_SHA1_Finish .....	292
20.2.28	HAL_HASH_MD5_Start_DMA .....	292
20.2.29	HAL_HASH_MD5_Finish .....	293
20.2.30	HAL_HMAC_MD5_Start_DMA.....	293
20.2.31	HAL_HMAC_SHA1_Start_DMA.....	293
20.2.32	HAL_HASH_GetState .....	294
20.2.33	HAL_HASH_IRQHandler.....	294
20.2.34	HAL_HASH_Init.....	294
20.2.35	HAL_HASH_DelInit .....	294
20.2.36	HAL_HASH_MspInit .....	294
20.2.37	HAL_HASH_MspDelInit .....	295
20.2.38	HAL_HASH_InCpltCallback .....	295
20.2.39	HAL_HASH_ErrorCallback.....	295
20.2.40	HAL_HASH_DgstCpltCallback.....	295
20.2.41	HAL_HASH_GetState .....	295
20.2.42	HAL_HASH_MspInit .....	296
20.2.43	HAL_HASH_MspDelInit .....	296
20.2.44	HAL_HASH_InCpltCallback .....	296
20.2.45	HAL_HASH_DgstCpltCallback.....	296
20.2.46	HAL_HASH_ErrorCallback.....	296
20.3	HASH Firmware driver defines.....	297
20.3.1	HASH.....	297
<b>21</b>	<b>HAL HCD Generic Driver.....</b>	<b>299</b>
21.1	HCD Firmware driver registers structures .....	299
21.1.1	HCD_HandleTypeDef.....	299
21.2	HCD Firmware driver API description .....	299
21.2.1	How to use this driver .....	299
21.2.2	Initialization and de-initialization functions .....	300
21.2.3	IO operation functions .....	300
21.2.4	Peripheral Control functions .....	300
21.2.5	Peripheral State functions .....	300
21.2.6	HAL_HCD_Init.....	300
21.2.7	HAL_HCD_HC_Init.....	301
21.2.8	HAL_HCD_HC_Halt .....	301
21.2.9	HAL_HCD_DelInit .....	301
21.2.10	HAL_HCD_MspInit .....	301

21.2.11	HAL_HCD_MspDeInit.....	302
21.2.12	HAL_HCD_HC_SubmitRequest.....	302
21.2.13	HAL_HCD_IRQHandler.....	302
21.2.14	HAL_HCD_SOF_Callback .....	302
21.2.15	HAL_HCD_Connect_Callback .....	303
21.2.16	HAL_HCD_Disconnect_Callback .....	303
21.2.17	HAL_HCD_HC_NotifyURBChange_Callback .....	303
21.2.18	HAL_HCD_Start .....	303
21.2.19	HAL_HCD_Stop .....	303
21.2.20	HAL_HCD_ResetPort.....	304
21.2.21	HAL_HCD_GetState.....	304
21.2.22	HAL_HCD_HC_GetURBState.....	304
21.2.23	HAL_HCD_HC_GetXferCount .....	304
21.2.24	HAL_HCD_HC_GetState .....	304
21.2.25	HAL_HCD_GetCurrentFrame .....	305
21.2.26	HAL_HCD_GetCurrentSpeed .....	305
21.3	HCD Firmware driver defines .....	305
21.3.1	HCD .....	305
<b>22</b>	<b>HAL I2C Generic Driver .....</b>	<b>307</b>
22.1	I2C Firmware driver registers structures .....	307
22.1.1	I2C_InitTypeDef.....	307
22.1.2	I2C_HandleTypeDef .....	307
22.2	I2C Firmware driver API description.....	308
22.2.1	How to use this driver .....	308
22.2.2	Initialization and de-initialization functions .....	311
22.2.3	IO operation functions .....	311
22.2.4	Peripheral State and Errors functions .....	313
22.2.5	HAL_I2C_Init .....	313
22.2.6	HAL_I2C_DeInit.....	313
22.2.7	HAL_I2C_MspInit .....	313
22.2.8	HAL_I2C_MspDeInit.....	313
22.2.9	HAL_I2C_Master_Transmit.....	314
22.2.10	HAL_I2C_Master_Receive.....	314
22.2.11	HAL_I2C_Slave_Transmit.....	314
22.2.12	HAL_I2C_Slave_Receive .....	315
22.2.13	HAL_I2C_Master_Transmit_IT.....	315
22.2.14	HAL_I2C_Master_Receive_IT.....	315
22.2.15	HAL_I2C_Slave_Transmit_IT.....	315

22.2.16	HAL_I2C_Slave_Receive_IT.....	316
22.2.17	HAL_I2C_Master_Transmit_DMA.....	316
22.2.18	HAL_I2C_Master_Receive_DMA.....	316
22.2.19	HAL_I2C_Slave_Transmit_DMA.....	316
22.2.20	HAL_I2C_Slave_Receive_DMA.....	317
22.2.21	HAL_I2C_Mem_Write.....	317
22.2.22	HAL_I2C_Mem_Read .....	317
22.2.23	HAL_I2C_Mem_Write_IT .....	318
22.2.24	HAL_I2C_Mem_Read_IT .....	318
22.2.25	HAL_I2C_Mem_Write_DMA .....	318
22.2.26	HAL_I2C_Mem_Read_DMA .....	319
22.2.27	HAL_I2C_IsDeviceReady.....	319
22.2.28	HAL_I2C_EV_IRQHandler .....	319
22.2.29	HAL_I2C_ER_IRQHandler .....	320
22.2.30	HAL_I2C_MasterTxCpltCallback.....	320
22.2.31	HAL_I2C_MasterRxCpltCallback .....	320
22.2.32	HAL_I2C_SlaveTxCpltCallback.....	320
22.2.33	HAL_I2C_SlaveRxCpltCallback .....	320
22.2.34	HAL_I2C_MemTxCpltCallback.....	320
22.2.35	HAL_I2C_MemRxCpltCallback .....	321
22.2.36	HAL_I2C_ErrorCallback .....	321
22.2.37	HAL_I2C_GetState .....	321
22.2.38	HAL_I2C_GetError .....	321
22.3	I2C Firmware driver defines .....	321
22.3.1	I2C .....	321
<b>23</b>	<b>HAL I2S Generic Driver .....</b>	<b>328</b>
23.1	I2S Firmware driver registers structures .....	328
23.1.1	I2S_InitTypeDef .....	328
23.1.2	I2S_HandleTypeDef .....	328
23.2	I2S Firmware driver API description.....	329
23.2.1	How to use this driver .....	329
23.2.2	Initialization and de-initialization functions .....	331
23.2.3	IO operation functions .....	331
23.2.4	Peripheral State and Errors functions .....	332
23.2.5	HAL_I2S_Init .....	332
23.2.6	HAL_I2S_DeInit.....	332
23.2.7	HAL_I2S_MspInit.....	333
23.2.8	HAL_I2S_MspDeInit .....	333

23.2.9	HAL_I2S_Transmit .....	333
23.2.10	HAL_I2S_Receive .....	333
23.2.11	HAL_I2S_Transmit_IT .....	334
23.2.12	HAL_I2S_Receive_IT .....	334
23.2.13	HAL_I2S_Transmit_DMA .....	335
23.2.14	HAL_I2S_Receive_DMA .....	335
23.2.15	HAL_I2S_DMAPause .....	336
23.2.16	HAL_I2S_DMAResume.....	336
23.2.17	HAL_I2S_DMAStop .....	336
23.2.18	HAL_I2S_IRQHandler .....	336
23.2.19	HAL_I2S_TxHalfCpltCallback .....	336
23.2.20	HAL_I2S_TxCpltCallback .....	336
23.2.21	HAL_I2S_RxHalfCpltCallback .....	337
23.2.22	HAL_I2S_RxCpltCallback .....	337
23.2.23	HAL_I2S_ErrorCallback .....	337
23.2.24	HAL_I2S_GetState .....	337
23.2.25	HAL_I2S_GetError .....	337
23.2.26	HAL_I2S_GetState .....	338
23.2.27	HAL_I2S_GetError .....	338
23.3	I2S Firmware driver defines .....	338
23.3.1	I2S .....	338
<b>24</b>	<b>HAL IRDA Generic Driver .....</b>	<b>343</b>
24.1	IRDA Firmware driver registers structures .....	343
24.1.1	IRDA_InitTypeDef.....	343
24.1.2	IRDA_HandleTypeDef .....	343
24.2	IRDA Firmware driver API description.....	344
24.2.1	How to use this driver .....	344
24.2.2	Initialization and Configuration functions.....	345
24.2.3	IO operation functions .....	346
24.2.4	Peripheral State and Errors functions .....	347
24.2.5	HAL_IRDA_Init .....	347
24.2.6	HAL_IRDA_DeInit.....	347
24.2.7	HAL_IRDA_MspInit .....	347
24.2.8	HAL_IRDA_MspDeInit.....	348
24.2.9	HAL_IRDA_Transmit.....	348
24.2.10	HAL_IRDA_Receive .....	348
24.2.11	HAL_IRDA_Transmit_IT .....	348
24.2.12	HAL_IRDA_Receive_IT.....	349

24.2.13	HAL_IRDA_Transmit_DMA .....	349
24.2.14	HAL_IRDA_Receive_DMA .....	349
24.2.15	HAL_IRDA_DMAPause .....	350
24.2.16	HAL_IRDA_DMAResume .....	350
24.2.17	HAL_IRDA_DMASStop .....	350
24.2.18	HAL_IRDA_IRQHandler .....	350
24.2.19	HAL_IRDA_TxCpltCallback .....	350
24.2.20	HAL_IRDA_TxHalfCpltCallback .....	351
24.2.21	HAL_IRDA_RxCpltCallback .....	351
24.2.22	HAL_IRDA_RxHalfCpltCallback .....	351
24.2.23	HAL_IRDA_ErrorCallback .....	351
24.2.24	HAL_IRDA_GetState .....	351
24.2.25	HAL_IRDA_GetError .....	352
24.3	IRDA Firmware driver defines .....	352
24.3.1	IRDA .....	352
<b>25</b>	<b>HAL IWDG Generic Driver .....</b>	<b>359</b>
25.1	IWDG Firmware driver registers structures .....	359
25.1.1	IWDG_InitTypeDef .....	359
25.1.2	IWDG_HandleTypeDef .....	359
25.2	IWDG Firmware driver API description .....	359
25.2.1	IWDG Specific features .....	359
25.2.2	How to use this driver .....	360
25.2.3	Initialization and de-initialization functions .....	360
25.2.4	IO operation functions .....	361
25.2.5	Peripheral State functions .....	361
25.2.6	HAL_IWDG_Init .....	361
25.2.7	HAL_IWDG_MspInit .....	361
25.2.8	HAL_IWDG_Start .....	361
25.2.9	HAL_IWDG_Refresh .....	362
25.2.10	HAL_IWDG_GetState .....	362
25.3	IWDG Firmware driver defines .....	362
25.3.1	IWDG .....	362
<b>26</b>	<b>HAL NAND Generic Driver .....</b>	<b>365</b>
26.1	NAND Firmware driver registers structures .....	365
26.1.1	NAND_IDTypeDef .....	365
26.1.2	NAND_AddressTypeDef .....	365
26.1.3	NAND_InfoTypeDef .....	365

26.1.4	NAND_HandleTypeDef .....	366
26.2	NAND Firmware driver API description .....	366
26.2.1	How to use this driver .....	366
26.2.2	NAND Initialization and de-initialization functions .....	367
26.2.3	NAND Input and Output functions .....	367
26.2.4	NAND Control functions .....	368
26.2.5	NAND State functions.....	368
26.2.6	HAL_NAND_Init.....	368
26.2.7	HAL_NAND_DelInit .....	368
26.2.8	HAL_NAND_MspInit.....	368
26.2.9	HAL_NAND_MspDelInit .....	369
26.2.10	HAL_NAND_IRQHandler .....	369
26.2.11	HAL_NAND_ITCallback .....	369
26.2.12	HAL_NAND_Read_ID .....	369
26.2.13	HAL_NAND_Reset .....	369
26.2.14	HAL_NAND_Read_Page .....	369
26.2.15	HAL_NAND_Write_Page.....	370
26.2.16	HAL_NAND_Read_SpareArea .....	370
26.2.17	HAL_NAND_Write_SpareArea.....	370
26.2.18	HAL_NAND_Erase_Block .....	371
26.2.19	HAL_NAND_Read_Status .....	371
26.2.20	HAL_NAND_Address_Inc .....	371
26.2.21	HAL_NAND_ECC_Enable .....	371
26.2.22	HAL_NAND_ECC_Disable.....	371
26.2.23	HAL_NAND_GetECC .....	372
26.2.24	HAL_NAND_GetState .....	372
26.2.25	HAL_NAND_Read_Status .....	372
26.3	NAND Firmware driver defines.....	372
26.3.1	NAND.....	372
27	HAL NOR Generic Driver.....	375
27.1	NOR Firmware driver registers structures .....	375
27.1.1	NOR_IDTypeDef .....	375
27.1.2	NOR_CFITypeDef .....	375
27.1.3	NOR_HandleTypeDef.....	375
27.2	NOR Firmware driver API description .....	376
27.2.1	How to use this driver .....	376
27.2.2	NOR Initialization and de_initialization functions .....	377
27.2.3	NOR Input and Output functions .....	377

27.2.4	NOR Control functions.....	377
27.2.5	NOR State functions.....	377
27.2.6	HAL_NOR_Init.....	377
27.2.7	HAL_NOR_Delnit .....	378
27.2.8	HAL_NOR_MsplInit .....	378
27.2.9	HAL_NOR_MspDelnit .....	378
27.2.10	HAL_NOR_MspWait.....	378
27.2.11	HAL_NOR_Read_ID .....	378
27.2.12	HAL_NOR_ReturnToReadMode.....	379
27.2.13	HAL_NOR_Read .....	379
27.2.14	HAL_NOR_Program.....	379
27.2.15	HAL_NOR_ReadBuffer .....	379
27.2.16	HAL_NOR_ProgramBuffer .....	379
27.2.17	HAL_NOR_Erase_Block .....	380
27.2.18	HAL_NOR_Erase_Chip.....	380
27.2.19	HAL_NOR_Read_CFI .....	380
27.2.20	HAL_NOR_WriteOperation_Enable .....	380
27.2.21	HAL_NOR_WriteOperation_Disable .....	381
27.2.22	HAL_NOR_GetState .....	381
27.2.23	HAL_NOR_GetStatus.....	381
27.3	NOR Firmware driver defines.....	381
27.3.1	NOR.....	381
<b>28</b>	<b>HAL PCCARD Generic Driver .....</b>	<b>384</b>
28.1	PCCARD Firmware driver registers structures.....	384
28.1.1	PCCARD_HandleTypeDef .....	384
28.2	PCCARD Firmware driver API description .....	384
28.2.1	How to use this driver .....	384
28.2.2	PCCARD Initialization and de-initialization functions .....	385
28.2.3	PCCARD Input and Output functions .....	385
28.2.4	PCCARD State functions.....	385
28.2.5	HAL_PCCARD_Init.....	385
28.2.6	HAL_PCCARD_Delnit.....	386
28.2.7	HAL_PCCARD_MsplInit.....	386
28.2.8	HAL_PCCARD_MspDelnit .....	386
28.2.9	HAL_PCCARD_Read_ID .....	386
28.2.10	HAL_PCCARD_Read_Sector .....	386
28.2.11	HAL_PCCARD_Write_Sector .....	387
28.2.12	HAL_PCCARD_Erase_Sector .....	387

28.2.13	HAL_PCCARD_Reset.....	387
28.2.14	HAL_PCCARD_IRQHandler .....	387
28.2.15	HAL_PCCARD_ITCallback .....	388
28.2.16	HAL_PCCARD_GetState .....	388
28.2.17	HAL_PCCARD_GetStatus .....	388
28.2.18	HAL_PCCARD_ReadStatus .....	388
28.3	PCCARD Firmware driver defines.....	389
28.3.1	PCCARD .....	389
<b>29</b>	<b>HAL PCD Generic Driver .....</b>	<b>391</b>
29.1	PCD Firmware driver registers structures .....	391
29.1.1	PCD_HandleTypeDef .....	391
29.2	PCD Firmware driver API description.....	391
29.2.1	How to use this driver .....	391
29.2.2	Initialization and de-initialization functions .....	392
29.2.3	IO operation functions .....	392
29.2.4	Peripheral Control functions .....	392
29.2.5	Peripheral State functions .....	393
29.2.6	HAL_PCD_Init .....	393
29.2.7	HAL_PCD_DelInit.....	393
29.2.8	HAL_PCD_MspInit .....	393
29.2.9	HAL_PCD_MspDelInit.....	393
29.2.10	HAL_PCD_Start .....	393
29.2.11	HAL_PCD_Stop.....	394
29.2.12	HAL_PCD_IRQHandler .....	394
29.2.13	HAL_PCD_DataOutStageCallback .....	394
29.2.14	HAL_PCD_DataInStageCallback .....	394
29.2.15	HAL_PCD_SetupStageCallback .....	394
29.2.16	HAL_PCD_SOFCallback.....	394
29.2.17	HAL_PCD_ResetCallback.....	395
29.2.18	HAL_PCD_SuspendCallback .....	395
29.2.19	HAL_PCD_ResumeCallback.....	395
29.2.20	HAL_PCD_ISOOUTIncompleteCallback.....	395
29.2.21	HAL_PCD_ISOINIncompleteCallback.....	395
29.2.22	HAL_PCD_ConnectCallback.....	396
29.2.23	HAL_PCD_DisconnectCallback .....	396
29.2.24	HAL_PCD_DevConnect .....	396
29.2.25	HAL_PCD_DevDisconnect .....	396
29.2.26	HAL_PCD_SetAddress .....	396

29.2.27	HAL_PCD_EP_Open .....	396
29.2.28	HAL_PCD_EP_Close .....	397
29.2.29	HAL_PCD_EP_Receive .....	397
29.2.30	HAL_PCD_EP_GetRxCount .....	397
29.2.31	HAL_PCD_EP_Transmit .....	397
29.2.32	HAL_PCD_EP_SetStall.....	398
29.2.33	HAL_PCD_EP_ClrStall.....	398
29.2.34	HAL_PCD_EP_Flush .....	398
29.2.35	HAL_PCD_ActivateRemoteWakeUp .....	398
29.2.36	HAL_PCD_DeActivateRemoteWakeUp.....	398
29.2.37	HAL_PCD_GetState.....	398
29.3	PCD Firmware driver defines .....	399
29.3.1	PCD .....	399
<b>30</b>	<b>HAL PCD Extension Driver .....</b>	<b>401</b>
30.1	PCDEx Firmware driver API description .....	401
30.1.1	Extended features functions .....	401
30.1.2	HAL_PCDEx_SetTxFiFo .....	401
30.1.3	HAL_PCDEx_SetRxFifo.....	401
<b>31</b>	<b>HAL PWR Generic Driver .....</b>	<b>402</b>
31.1	PWR Firmware driver registers structures .....	402
31.1.1	PWR_PVDTTypeDef .....	402
31.2	PWR Firmware driver API description .....	402
31.2.1	Initialization and de-initialization functions .....	402
31.2.2	Peripheral Control functions .....	402
31.2.3	HAL_PWR_DelInit.....	404
31.2.4	HAL_PWR_EnableBkUpAccess .....	404
31.2.5	HAL_PWR_DisableBkUpAccess.....	405
31.2.6	HAL_PWR_ConfigPVD .....	405
31.2.7	HAL_PWR_EnablePVD.....	405
31.2.8	HAL_PWR_DisablePVD.....	405
31.2.9	HAL_PWR_EnableWakeUpPin.....	405
31.2.10	HAL_PWR_DisableWakeUpPin .....	405
31.2.11	HAL_PWR_EnterSLEEPMode.....	406
31.2.12	HAL_PWR_EnterSTOPMode.....	406
31.2.13	HAL_PWR_EnterSTANDBYMode .....	407
31.2.14	HAL_PWR_PVD_IRQHandler.....	407
31.2.15	HAL_PWR_PVDCallback.....	407

31.2.16	HAL_PWR_EnableSleepOnExit.....	407
31.2.17	HAL_PWR_DisableSleepOnExit.....	407
31.2.18	HAL_PWR_EnableSEVOnPend .....	408
31.2.19	HAL_PWR_DisableSEVOnPend.....	408
31.3	PWR Firmware driver defines .....	408
31.3.1	PWR .....	408
<b>32</b>	<b>HAL PWR Extension Driver .....</b>	<b>413</b>
32.1	PWREx Firmware driver API description.....	413
32.1.1	Peripheral extended features functions.....	413
32.1.2	HAL_PWREx_EnableBkUpReg .....	413
32.1.3	HAL_PWREx_DisableBkUpReg .....	413
32.1.4	HAL_PWREx_EnableFlashPowerDown .....	414
32.1.5	HAL_PWREx_DisableFlashPowerDown.....	414
32.2	PWREx Firmware driver defines .....	414
32.2.1	PWREx .....	414
<b>33</b>	<b>HAL RCC Generic Driver.....</b>	<b>415</b>
33.1	RCC Firmware driver registers structures .....	415
33.1.1	RCC_PLLInitTypeDef .....	415
33.1.2	RCC_OscInitTypeDef .....	415
33.1.3	RCC_ClkInitTypeDef .....	416
33.2	RCC Firmware driver API description .....	417
33.2.1	RCC specific features.....	417
33.2.2	RCC Limitations.....	417
33.2.3	Initialization and de-initialization functions .....	417
33.2.4	Peripheral Control functions .....	419
33.2.5	HAL_RCC_DeInit .....	419
33.2.6	HAL_RCC_OscConfig .....	419
33.2.7	HAL_RCC_ClockConfig .....	420
33.2.8	HAL_RCC_MCOConfig .....	420
33.2.9	HAL_RCC_EnableCSS .....	421
33.2.10	HAL_RCC_DisableCSS .....	421
33.2.11	HAL_RCC_GetSysClockFreq .....	421
33.2.12	HAL_RCC_GetHCLKFreq.....	422
33.2.13	HAL_RCC_GetPCLK1Freq .....	422
33.2.14	HAL_RCC_GetPCLK2Freq .....	422
33.2.15	HAL_RCC_GetOscConfig .....	422
33.2.16	HAL_RCC_GetClockConfig .....	423
33.2.17	HAL_RCC_NMI_IRQHandler .....	423

33.2.18	HAL_RCC_CSSCallback.....	423
33.3	RCC Firmware driver defines .....	423
33.3.1	RCC .....	423
<b>34</b>	<b>HAL RCC Extension Driver .....</b>	<b>448</b>
34.1	RCCEEx Firmware driver registers structures .....	448
34.1.1	RCC_PLLI2SInitTypeDef.....	448
34.1.2	RCC_PeriphCLKInitTypeDef.....	448
34.2	RCCEEx Firmware driver API description .....	449
34.2.1	Extended Peripheral Control functions .....	449
34.2.2	HAL_RCCEEx_PeriphCLKConfig.....	449
34.2.3	HAL_RCCEEx_GetPeriphCLKConfig .....	449
34.3	RCCEEx Firmware driver defines .....	449
34.3.1	RCCEx .....	449
<b>35</b>	<b>HAL RNG Generic Driver.....</b>	<b>453</b>
35.1	RNG Firmware driver registers structures .....	453
35.1.1	RNG_HandleTypeDef.....	453
35.2	RNG Firmware driver API description .....	453
35.2.1	How to use this driver .....	453
35.2.2	Initialization and de-initialization functions .....	453
35.2.3	Peripheral Control functions .....	454
35.2.4	Peripheral State functions .....	454
35.2.5	HAL_RNG_Init.....	454
35.2.6	HAL_RNG_Delinit .....	454
35.2.7	HAL_RNG_MspInit .....	454
35.2.8	HAL_RNG_MspDelInit .....	455
35.2.9	HAL_RNG_GenerateRandomNumber .....	455
35.2.10	HAL_RNG_GenerateRandomNumber_IT .....	455
35.2.11	HAL_RNG_IRQHandler.....	455
35.2.12	HAL_RNG_GetRandomNumber .....	456
35.2.13	HAL_RNG_GetRandomNumber_IT .....	456
35.2.14	HAL_RNG_ReadLastRandomNumber.....	456
35.2.15	HAL_RNG_ReadyDataCallback.....	456
35.2.16	HAL_RNG_ErrorCallback.....	457
35.2.17	HAL_RNG_GetState .....	457
35.3	RNG Firmware driver defines .....	457
35.3.1	RNG .....	457
<b>36</b>	<b>HAL RTC Generic Driver .....</b>	<b>461</b>

Contents	UM1940
36.1 RTC Firmware driver registers structures .....	461
36.1.1 RTC_InitTypeDef.....	461
36.1.2 RTC_TimeTypeDef.....	461
36.1.3 RTC_DateTypeDef .....	462
36.1.4 RTC_AlarmTypeDef .....	462
36.1.5 RTC_HandleTypeDef .....	463
36.2 RTC Firmware driver API description.....	464
36.2.1 Backup Domain Operating Condition .....	464
36.2.2 Backup Domain Reset.....	464
36.2.3 Backup Domain Access.....	464
36.2.4 How to use this driver .....	464
36.2.5 RTC and low power modes .....	465
36.2.6 Initialization and de-initialization functions .....	465
36.2.7 RTC Time and Date functions .....	466
36.2.8 RTC Alarm functions .....	466
36.2.9 Peripheral Control functions .....	466
36.2.10 Peripheral State functions .....	466
36.2.11 HAL_RTC_Init .....	466
36.2.12 HAL_RTC_DelInit.....	467
36.2.13 HAL_RTC_MspInit.....	467
36.2.14 HAL_RTC_MspDeInit .....	467
36.2.15 HAL_RTC_SetTime.....	467
36.2.16 HAL_RTC_GetTime .....	467
36.2.17 HAL_RTC_SetDate .....	468
36.2.18 HAL_RTC_GetDate .....	468
36.2.19 HAL_RTC_SetAlarm .....	469
36.2.20 HAL_RTC_SetAlarm_IT .....	469
36.2.21 HAL_RTC_DeactivateAlarm.....	469
36.2.22 HAL_RTC_GetAlarm .....	469
36.2.23 HAL_RTC_AlarmIRQHandler.....	470
36.2.24 HAL_RTC_AlarmAEventCallback .....	470
36.2.25 HAL_RTC_PollForAlarmAEvent.....	470
36.2.26 HAL_RTC_WaitForSynchro .....	470
36.2.27 HAL_RTC_GetState .....	471
36.3 RTC Firmware driver defines .....	471
36.3.1 RTC .....	471
<b>37 HAL RTC Extension Driver .....</b>	<b>480</b>
37.1 RTCEx Firmware driver registers structures .....	480

37.1.1	RTC_TamperTypeDef .....	480
37.2	RTCEEx Firmware driver API description.....	480
37.2.1	How to use this driver .....	480
37.2.2	RTC TimeStamp and Tamper functions .....	481
37.2.3	RTC Wake-up functions .....	481
37.2.4	Extension Peripheral Control functions .....	482
37.2.5	Extended features functions .....	482
37.2.6	HAL_RTCEEx_SetTimeStamp .....	482
37.2.7	HAL_RTCEEx_SetTimeStamp_IT .....	483
37.2.8	HAL_RTCEEx_DeactivateTimeStamp .....	483
37.2.9	HAL_RTCEEx_GetTimeStamp.....	483
37.2.10	HAL_RTCEEx_SetTamper .....	484
37.2.11	HAL_RTCEEx_SetTamper_IT .....	484
37.2.12	HAL_RTCEEx_DeactivateTamper .....	484
37.2.13	HAL_RTCEEx_TamperTimeStampIRQHandler .....	484
37.2.14	HAL_RTCEEx_TimeStampEventCallback .....	485
37.2.15	HAL_RTCEEx_Tamper1EventCallback .....	485
37.2.16	HAL_RTCEEx_PollForTimeStampEvent.....	485
37.2.17	HAL_RTCEEx_PollForTamper1Event .....	485
37.2.18	HAL_RTCEEx_SetWakeUpTimer .....	485
37.2.19	HAL_RTCEEx_SetWakeUpTimer_IT .....	486
37.2.20	HAL_RTCEEx_DeactivateWakeUpTimer.....	486
37.2.21	HAL_RTCEEx_GetWakeUpTimer .....	486
37.2.22	HAL_RTCEEx_WakeUpTimerIRQHandler .....	486
37.2.23	HAL_RTCEEx_WakeUpTimerEventCallback .....	486
37.2.24	HAL_RTCEEx_PollForWakeUpTimerEvent .....	487
37.2.25	HAL_RTCEEx_BKUPWrite .....	487
37.2.26	HAL_RTCEEx_BKUPRead .....	487
37.2.27	HAL_RTCEEx_SetCoarseCalib.....	487
37.2.28	HAL_RTCEEx_DeactivateCoarseCalib .....	488
37.2.29	HAL_RTCEEx_SetCalibrationOutPut .....	488
37.2.30	HAL_RTCEEx_DeactivateCalibrationOutPut .....	488
37.2.31	HAL_RTCEEx_SetRefClock .....	488
37.2.32	HAL_RTCEEx_DeactivateRefClock .....	489
37.2.33	HAL_RTCEEx_AlarmBEventCallback .....	489
37.2.34	HAL_RTCEEx_PollForAlarmBEvent .....	489
37.3	RTCEEx Firmware driver defines .....	489
37.3.1	RTCEEx .....	489

---

<b>38 HAL SD Generic Driver .....</b>	<b>503</b>
38.1 SD Firmware driver registers structures .....	503
38.1.1 SD_HandleTypeDef.....	503
38.1.2 HAL_SD_CSDTypedef.....	504
38.1.3 HAL_SD_CIDTypedef .....	506
38.1.4 HAL_SD_CardStatusTypedef .....	507
38.1.5 HAL_SD_CardInfoTypedef.....	507
38.2 SD Firmware driver API description .....	508
38.2.1 How to use this driver .....	508
38.2.2 Initialization and de-initialization functions .....	510
38.2.3 IO operation functions .....	510
38.2.4 Peripheral Control functions .....	510
38.2.5 Peripheral State functions .....	511
38.2.6 HAL_SD_Init.....	511
38.2.7 HAL_SD_DelInit .....	511
38.2.8 HAL_SD_MsplInit.....	511
38.2.9 HAL_SD_MspDelInit .....	511
38.2.10 HAL_SD_ReadBlocks .....	512
38.2.11 HAL_SD_WriteBlocks.....	512
38.2.12 HAL_SD_ReadBlocks_DMA .....	512
38.2.13 HAL_SD_WriteBlocks_DMA .....	513
38.2.14 HAL_SD_CheckReadOperation.....	513
38.2.15 HAL_SD_CheckWriteOperation .....	513
38.2.16 HAL_SD_Erase .....	513
38.2.17 HAL_SD_IRQHandler.....	513
38.2.18 HAL_SD_XferCpltCallback.....	514
38.2.19 HAL_SD_XferErrorCallback .....	514
38.2.20 HAL_SD_DMA_RxCpltCallback.....	514
38.2.21 HAL_SD_DMA_RxErrorCallback .....	514
38.2.22 HAL_SD_DMA_TxCpltCallback .....	514
38.2.23 HAL_SD_DMA_TxErrorCallback .....	515
38.2.24 HAL_SD_Get_CardInfo .....	515
38.2.25 HAL_SD_WideBusOperation_Config.....	515
38.2.26 HAL_SD_StopTransfer.....	515
38.2.27 HAL_SD_HighSpeed.....	516
38.2.28 HAL_SD_SendSDStatus.....	516
38.2.29 HAL_SD_GetStatus.....	516
38.2.30 HAL_SD_GetCardStatus.....	516

38.3	SD Firmware driver defines .....	516
38.3.1	SD.....	516
<b>39</b>	<b>HAL SMARTCARD Generic Driver.....</b>	<b>529</b>
39.1	SMARTCARD Firmware driver registers structures .....	529
39.1.1	SMARTCARD_InitTypeDef .....	529
39.1.2	SMARTCARD_HandleTypeDef.....	530
39.2	SMARTCARD Firmware driver API description.....	531
39.2.1	How to use this driver .....	531
39.2.2	Initialization and Configuration functions .....	532
39.2.3	IO operation functions .....	533
39.2.4	Peripheral State and Errors functions .....	535
39.2.5	HAL_SMARTCARD_Init .....	535
39.2.6	HAL_SMARTCARD_DelInit .....	535
39.2.7	HAL_SMARTCARD_MspInit .....	535
39.2.8	HAL_SMARTCARD_MspDelInit .....	536
39.2.9	HAL_SMARTCARD_ReInit .....	536
39.2.10	HAL_SMARTCARD_Transmit.....	536
39.2.11	HAL_SMARTCARD_Receive.....	536
39.2.12	HAL_SMARTCARD_Transmit_IT .....	537
39.2.13	HAL_SMARTCARD_Receive_IT .....	537
39.2.14	HAL_SMARTCARD_Transmit_DMA.....	537
39.2.15	HAL_SMARTCARD_Receive_DMA.....	537
39.2.16	HAL_SMARTCARD_IRQHandler.....	538
39.2.17	HAL_SMARTCARD_TxCpltCallback .....	538
39.2.18	HAL_SMARTCARD_RxCpltCallback .....	538
39.2.19	HAL_SMARTCARD_ErrorCallback .....	538
39.2.20	HAL_SMARTCARD_GetState .....	539
39.2.21	HAL_SMARTCARD_GetError .....	539
39.3	SMARTCARD Firmware driver defines .....	539
39.3.1	SMARTCARD.....	539
<b>40</b>	<b>HAL SPI Generic Driver.....</b>	<b>548</b>
40.1	SPI Firmware driver registers structures .....	548
40.1.1	SPI_InitTypeDef .....	548
40.1.2	__SPI_HandleTypeDef.....	549
40.2	SPI Firmware driver API description .....	550
40.2.1	How to use this driver .....	550
40.2.2	Initialization and de-initialization functions .....	550

40.2.3	IO operation functions .....	551
40.2.4	Peripheral State and Errors functions .....	551
40.2.5	HAL_SPI_Init .....	552
40.2.6	HAL_SPI_DeInit .....	552
40.2.7	HAL_SPI_MspInit .....	552
40.2.8	HAL_SPI_MspDeInit.....	552
40.2.9	HAL_SPI_Transmit.....	552
40.2.10	HAL_SPI_Receive.....	553
40.2.11	HAL_SPI_TransmitReceive.....	553
40.2.12	HAL_SPI_Transmit_IT.....	553
40.2.13	HAL_SPI_Receive_IT.....	553
40.2.14	HAL_SPI_TransmitReceive_IT .....	554
40.2.15	HAL_SPI_Transmit_DMA.....	554
40.2.16	HAL_SPI_Receive_DMA.....	554
40.2.17	HAL_SPI_TransmitReceive_DMA.....	554
40.2.18	HAL_SPI_DMAPause.....	555
40.2.19	HAL_SPI_DMAResume .....	555
40.2.20	HAL_SPI_DMAStop .....	555
40.2.21	HAL_SPI_IRQHandler.....	555
40.2.22	HAL_SPI_TxCpltCallback .....	556
40.2.23	HAL_SPI_RxCpltCallback .....	556
40.2.24	HAL_SPI_TxRxCpltCallback .....	556
40.2.25	HAL_SPI_TxHalfCpltCallback .....	556
40.2.26	HAL_SPI_RxHalfCpltCallback.....	556
40.2.27	HAL_SPI_TxRxHalfCpltCallback.....	556
40.2.28	HAL_SPI_ErrorCallback .....	557
40.2.29	HAL_SPI_GetState.....	557
40.2.30	HAL_SPI_GetError .....	557
40.3	SPI Firmware driver defines .....	557
40.3.1	SPI .....	557
<b>41</b>	<b>HAL SRAM Generic Driver .....</b>	<b>563</b>
41.1	SRAM Firmware driver registers structures.....	563
41.1.1	SRAM_HandleTypeDef .....	563
41.2	SRAM Firmware driver API description .....	563
41.2.1	How to use this driver .....	563
41.2.2	SRAM Initialization and de_initialization functions .....	564
41.2.3	SRAM Input and Output functions .....	564
41.2.4	SRAM Control functions .....	564

41.2.5	SRAM State functions .....	565
41.2.6	HAL_SRAM_Init .....	565
41.2.7	HAL_SRAM_DelInit.....	565
41.2.8	HAL_SRAM_MspInit.....	565
41.2.9	HAL_SRAM_MspDeInit.....	565
41.2.10	HAL_SRAM_DMA_XferCpltCallback .....	566
41.2.11	HAL_SRAM_DMA_XferErrorCallback.....	566
41.2.12	HAL_SRAM_Read_8b.....	566
41.2.13	HAL_SRAM_Write_8b.....	566
41.2.14	HAL_SRAM_Read_16b.....	566
41.2.15	HAL_SRAM_Write_16b.....	567
41.2.16	HAL_SRAM_Read_32b.....	567
41.2.17	HAL_SRAM_Write_32b.....	567
41.2.18	HAL_SRAM_Read_DMA.....	568
41.2.19	HAL_SRAM_Write_DMA.....	568
41.2.20	HAL_SRAM_WriteOperation_Enable.....	568
41.2.21	HAL_SRAM_WriteOperation_Disable.....	568
41.2.22	HAL_SRAM_GetState .....	568
41.3	SRAM Firmware driver defines .....	569
41.3.1	SRAM .....	569
<b>42</b>	<b>HAL TIM Generic Driver .....</b>	<b>570</b>
42.1	TIM Firmware driver registers structures.....	570
42.1.1	TIM_Base_InitTypeDef.....	570
42.1.2	TIM_OC_InitTypeDef.....	570
42.1.3	TIM_OnePulse_InitTypeDef .....	571
42.1.4	TIM_IC_InitTypeDef .....	572
42.1.5	TIM_Encoder_InitTypeDef .....	572
42.1.6	TIM_ClockConfigTypeDef .....	573
42.1.7	TIM_ClearInputConfigTypeDef.....	574
42.1.8	TIM_SlaveConfigTypeDef .....	574
42.1.9	TIM_HandleTypeDef .....	575
42.2	TIM Firmware driver API description .....	575
42.2.1	TIMER Generic features.....	575
42.2.2	How to use this driver.....	576
42.2.3	Time Base functions .....	576
42.2.4	Time Output Compare functions .....	577
42.2.5	Time PWM functions .....	577
42.2.6	Time Input Capture functions .....	578

---

42.2.7	Time One Pulse functions .....	578
42.2.8	Time Encoder functions.....	579
42.2.9	IRQ handler management.....	579
42.2.10	Peripheral Control functions .....	579
42.2.11	TIM Callbacks functions .....	580
42.2.12	Peripheral State functions .....	580
42.2.13	HAL_TIM_Base_Init .....	580
42.2.14	HAL_TIM_Base_DelInit.....	581
42.2.15	HAL_TIM_Base_MspInit.....	581
42.2.16	HAL_TIM_Base_MspDelInit.....	581
42.2.17	HAL_TIM_Base_Start.....	581
42.2.18	HAL_TIM_Base_Stop.....	581
42.2.19	HAL_TIM_Base_Start_IT .....	581
42.2.20	HAL_TIM_Base_Stop_IT.....	582
42.2.21	HAL_TIM_Base_Start_DMA .....	582
42.2.22	HAL_TIM_Base_Stop_DMA.....	582
42.2.23	HAL_TIM_OC_Init .....	582
42.2.24	HAL_TIM_OC_DelInit.....	583
42.2.25	HAL_TIM_OC_MspInit .....	583
42.2.26	HAL_TIM_OC_MspDelInit.....	583
42.2.27	HAL_TIM_OC_Start .....	583
42.2.28	HAL_TIM_OC_Stop.....	583
42.2.29	HAL_TIM_OC_Start_IT .....	584
42.2.30	HAL_TIM_OC_Stop_IT .....	584
42.2.31	HAL_TIM_OC_Start_DMA .....	584
42.2.32	HAL_TIM_OC_Stop_DMA .....	585
42.2.33	HAL_TIM_PWM_Init.....	585
42.2.34	HAL_TIM_PWM_DelInit .....	585
42.2.35	HAL_TIM_PWM_MspInit .....	585
42.2.36	HAL_TIM_PWM_MspDelInit .....	585
42.2.37	HAL_TIM_PWM_Start.....	586
42.2.38	HAL_TIM_PWM_Stop .....	586
42.2.39	HAL_TIM_PWM_Start_IT .....	586
42.2.40	HAL_TIM_PWM_Stop_IT .....	587
42.2.41	HAL_TIM_PWM_Start_DMA .....	587
42.2.42	HAL_TIM_PWM_Stop_DMA .....	587
42.2.43	HAL_TIM_IC_Init .....	587
42.2.44	HAL_TIM_IC_DelInit .....	588
42.2.45	HAL_TIM_IC_MspInit .....	588

42.2.46	HAL_TIM_IC_MspDeInit.....	588
42.2.47	HAL_TIM_IC_Start .....	588
42.2.48	HAL_TIM_IC_Stop .....	588
42.2.49	HAL_TIM_IC_Start_IT .....	589
42.2.50	HAL_TIM_IC_Stop_IT .....	589
42.2.51	HAL_TIM_IC_Start_DMA .....	589
42.2.52	HAL_TIM_IC_Stop_DMA .....	590
42.2.53	HAL_TIM_OnePulse_Init.....	590
42.2.54	HAL_TIM_OnePulse_DeInit .....	590
42.2.55	HAL_TIM_OnePulse_MspInit.....	590
42.2.56	HAL_TIM_OnePulse_MspDeInit .....	591
42.2.57	HAL_TIM_OnePulse_Start.....	591
42.2.58	HAL_TIM_OnePulse_Stop .....	591
42.2.59	HAL_TIM_OnePulse_Start_IT.....	591
42.2.60	HAL_TIM_OnePulse_Stop_IT .....	592
42.2.61	HAL_TIM_Encoder_Init.....	592
42.2.62	HAL_TIM_Encoder_DeInit .....	592
42.2.63	HAL_TIM_Encoder_MspInit .....	592
42.2.64	HAL_TIM_Encoder_MspDeInit.....	593
42.2.65	HAL_TIM_Encoder_Start .....	593
42.2.66	HAL_TIM_Encoder_Stop .....	593
42.2.67	HAL_TIM_Encoder_Start_IT .....	593
42.2.68	HAL_TIM_Encoder_Stop_IT .....	594
42.2.69	HAL_TIM_Encoder_Start_DMA .....	594
42.2.70	HAL_TIM_Encoder_Stop_DMA .....	594
42.2.71	HAL_TIM_IRQHandler .....	595
42.2.72	HAL_TIM_OC_ConfigChannel .....	595
42.2.73	HAL_TIM_IC_ConfigChannel.....	595
42.2.74	HAL_TIM_PWM_ConfigChannel.....	595
42.2.75	HAL_TIM_OnePulse_ConfigChannel.....	596
42.2.76	HAL_TIM_DMABurst_WriteStart .....	596
42.2.77	HAL_TIM_DMABurst_WriteStop .....	597
42.2.78	HAL_TIM_DMABurst_ReadStart .....	597
42.2.79	HAL_TIM_DMABurst_ReadStop .....	598
42.2.80	HAL_TIM_GenerateEvent .....	598
42.2.81	HAL_TIM_ConfigOCrefClear.....	599
42.2.82	HAL_TIM_ConfigClockSource .....	599
42.2.83	HAL_TIM_ConfigTI1Input.....	599

42.2.84	HAL_TIM_SlaveConfigSynchronization .....	600
42.2.85	HAL_TIM_SlaveConfigSynchronization_IT .....	600
42.2.86	HAL_TIM_ReadCapturedValue.....	600
42.2.87	HAL_TIM_PeriodElapsedCallback .....	601
42.2.88	HAL_TIM_OC_DelayElapsedCallback.....	601
42.2.89	HAL_TIM_IC_CaptureCallback .....	601
42.2.90	HAL_TIM_PWM_PulseFinishedCallback .....	601
42.2.91	HAL_TIM_TriggerCallback .....	601
42.2.92	HAL_TIM_ErrorCallback.....	602
42.2.93	HAL_TIM_Base_GetState.....	602
42.2.94	HAL_TIM_OC_GetState.....	602
42.2.95	HAL_TIM_PWM_GetState .....	602
42.2.96	HAL_TIM_IC_GetState.....	602
42.2.97	HAL_TIM_OnePulse_GetState .....	602
42.2.98	HAL_TIM_Encoder_GetState.....	603
42.3	TIM Firmware driver defines.....	603
42.3.1	TIM.....	603
<b>43</b>	<b>HAL TIM Extension Driver.....</b>	<b>618</b>
43.1	TIMEx Firmware driver registers structures.....	618
43.1.1	TIM_HallSensor_InitTypeDef .....	618
43.1.2	TIM_MasterConfigTypeDef .....	618
43.1.3	TIM_BreakDeadTimeConfigTypeDef .....	618
43.2	TIMEx Firmware driver API description.....	619
43.2.1	TIMER Extended features .....	619
43.2.2	How to use this driver .....	619
43.2.3	Timer Hall Sensor functions .....	620
43.2.4	Timer Complementary Output Compare functions.....	621
43.2.5	Timer Complementary PWM functions.....	621
43.2.6	Timer Complementary One Pulse functions.....	621
43.2.7	Peripheral Control functions .....	622
43.2.8	Extension Callbacks functions.....	622
43.2.9	Extension Peripheral State functions .....	622
43.2.10	HAL_TIMEx_HallSensor_Init.....	622
43.2.11	HAL_TIMEx_HallSensor_Delinit .....	623
43.2.12	HAL_TIMEx_HallSensor_MspInit.....	623
43.2.13	HAL_TIMEx_HallSensor_MspDelinit .....	623
43.2.14	HAL_TIMEx_HallSensor_Start.....	623
43.2.15	HAL_TIMEx_HallSensor_Stop .....	623

43.2.16	HAL_TIMEx_HallSensor_Start_IT .....	624
43.2.17	HAL_TIMEx_HallSensor_Stop_IT .....	624
43.2.18	HAL_TIMEx_HallSensor_Start_DMA .....	624
43.2.19	HAL_TIMEx_HallSensor_Stop_DMA .....	624
43.2.20	HAL_TIMEx_OCN_Start .....	624
43.2.21	HAL_TIMEx_OCN_Stop .....	625
43.2.22	HAL_TIMEx_OCN_Start_IT .....	625
43.2.23	HAL_TIMEx_OCN_Stop_IT .....	625
43.2.24	HAL_TIMEx_OCN_Start_DMA .....	626
43.2.25	HAL_TIMEx_OCN_Stop_DMA .....	626
43.2.26	HAL_TIMEx_PWMN_Start .....	626
43.2.27	HAL_TIMEx_PWMN_Stop .....	627
43.2.28	HAL_TIMEx_PWMN_Start_IT .....	627
43.2.29	HAL_TIMEx_PWMN_Stop_IT .....	627
43.2.30	HAL_TIMEx_PWMN_Start_DMA .....	628
43.2.31	HAL_TIMEx_PWMN_Stop_DMA .....	628
43.2.32	HAL_TIMEx_OnePulseN_Start .....	628
43.2.33	HAL_TIMEx_OnePulseN_Stop .....	628
43.2.34	HAL_TIMEx_OnePulseN_Start_IT .....	629
43.2.35	HAL_TIMEx_OnePulseN_Stop_IT .....	629
43.2.36	HAL_TIMEx_ConfigCommutationEvent .....	629
43.2.37	HAL_TIMEx_ConfigCommutationEvent_IT .....	630
43.2.38	HAL_TIMEx_ConfigCommutationEvent_DMA .....	631
43.2.39	HAL_TIMEx_MasterConfigSynchronization .....	631
43.2.40	HAL_TIMEx_ConfigBreakDeadTime .....	631
43.2.41	HAL_TIMEx_RemapConfig .....	632
43.2.42	HAL_TIMEx_ChamutationCallback .....	632
43.2.43	HAL_TIMEx_BreakCallback .....	632
43.2.44	TIMEEx_DMACommutationCplt .....	633
43.2.45	HAL_TIMEx_HallSensor_GetState .....	633
43.3	TIMEx Firmware driver defines .....	633
43.3.1	TIMEx .....	633
<b>44</b>	<b>HAL UART Generic Driver.....</b>	<b>635</b>
44.1	UART Firmware driver registers structures .....	635
44.1.1	UART_InitTypeDef .....	635
44.1.2	UART_HandleTypeDef .....	635
44.2	UART Firmware driver API description .....	636
44.2.1	How to use this driver .....	636

44.2.2	Initialization and Configuration functions .....	638
44.2.3	IO operation functions .....	639
44.2.4	Peripheral Control functions .....	640
44.2.5	Peripheral State and Errors functions .....	640
44.2.6	HAL_UART_Init .....	640
44.2.7	HAL_HalfDuplex_Init .....	641
44.2.8	HAL_LIN_Init .....	641
44.2.9	HAL_MultiProcessor_Init .....	641
44.2.10	HAL_UART_DelInit .....	642
44.2.11	HAL_UART_MspInit .....	642
44.2.12	HAL_UART_MspDelInit .....	642
44.2.13	HAL_UART_Transmit .....	642
44.2.14	HAL_UART_Receive .....	643
44.2.15	HAL_UART_Transmit_IT .....	643
44.2.16	HAL_UART_Receive_IT .....	643
44.2.17	HAL_UART_Transmit_DMA .....	643
44.2.18	HAL_UART_Receive_DMA .....	644
44.2.19	HAL_UART_DMAPause .....	644
44.2.20	HAL_UART_DMAResume .....	644
44.2.21	HAL_UART_DMAStop .....	644
44.2.22	HAL_UART_IRQHandler .....	645
44.2.23	HAL_UART_TxCpltCallback .....	645
44.2.24	HAL_UART_TxHalfCpltCallback .....	645
44.2.25	HAL_UART_RxCpltCallback .....	645
44.2.26	HAL_UART_RxHalfCpltCallback .....	645
44.2.27	HAL_UART_ErrorCallback .....	646
44.2.28	HAL_LIN_SendBreak .....	646
44.2.29	HAL_MultiProcessor_EnterMuteMode .....	646
44.2.30	HAL_MultiProcessor_ExitMuteMode .....	646
44.2.31	HAL_HalfDuplex_EnableTransmitter .....	646
44.2.32	HAL_HalfDuplex_EnableReceiver .....	647
44.2.33	HAL_UART_GetState .....	647
44.2.34	HAL_UART_GetError .....	647
44.3	UART Firmware driver defines .....	647
44.3.1	UART .....	647
<b>45</b>	<b>HAL USART Generic Driver .....</b>	<b>659</b>
45.1	USART Firmware driver registers structures .....	659
45.1.1	USART_InitTypeDef .....	659

45.1.2	USART_HandleTypeDef .....	660
45.2	USART Firmware driver API description .....	660
45.2.1	How to use this driver .....	660
45.2.2	Initialization and Configuration functions .....	662
45.2.3	IO operation functions .....	663
45.2.4	Peripheral State and Errors functions .....	664
45.2.5	HAL_USART_Init.....	664
45.2.6	HAL_USART_Delnit .....	664
45.2.7	HAL_USART_MsplInit.....	664
45.2.8	HAL_USART_MspDelnit .....	665
45.2.9	HAL_USART_Transmit .....	665
45.2.10	HAL_USART_Receive .....	665
45.2.11	HAL_USART_TransmitReceive .....	665
45.2.12	HAL_USART_Transmit_IT .....	666
45.2.13	HAL_USART_Receive_IT .....	666
45.2.14	HAL_USART_TransmitReceive_IT .....	666
45.2.15	HAL_USART_Transmit_DMA .....	667
45.2.16	HAL_USART_Receive_DMA .....	667
45.2.17	HAL_USART_TransmitReceive_DMA .....	667
45.2.18	HAL_USART_DMAPause .....	668
45.2.19	HAL_USART_DMAResume .....	668
45.2.20	HAL_USART_DMAStop .....	668
45.2.21	HAL_USART_IRQHandler .....	668
45.2.22	HAL_USART_TxCpltCallback .....	668
45.2.23	HAL_USART_TxHalfCpltCallback .....	669
45.2.24	HAL_USART_RxCpltCallback.....	669
45.2.25	HAL_USART_RxHalfCpltCallback .....	669
45.2.26	HAL_USART_TxRxCpltCallback .....	669
45.2.27	HAL_USART_ErrorCallback .....	669
45.2.28	HAL_USART_GetState .....	670
45.2.29	HAL_USART_GetError.....	670
45.3	USART Firmware driver defines.....	670
45.3.1	USART.....	670
46	HAL WWDG Generic Driver .....	678
46.1	WWDG Firmware driver registers structures.....	678
46.1.1	WWDG_InitTypeDef .....	678
46.1.2	WWDG_HandleTypeDef .....	678
46.2	WWDG Firmware driver API description .....	679

---

46.2.1	WWDG specific features .....	679
46.2.2	How to use this driver .....	679
46.2.3	Initialization and de-initialization functions .....	679
46.2.4	IO operation functions .....	680
46.2.5	Peripheral State functions .....	680
46.2.6	HAL_WWDG_Init.....	680
46.2.7	HAL_WWDG_DelInit.....	680
46.2.8	HAL_WWDG_MspInit.....	681
46.2.9	HAL_WWDG_MspDelInit .....	681
46.2.10	HAL_WWDG_WakeupCallback .....	681
46.2.11	HAL_WWDG_Start.....	681
46.2.12	HAL_WWDG_Start_IT.....	681
46.2.13	HAL_WWDG_Refresh.....	682
46.2.14	HAL_WWDG_IRQHandler .....	682
46.2.15	HAL_WWDG_WakeupCallback .....	682
46.2.16	HAL_WWDG_GetState .....	682
46.3	WWDG Firmware driver defines.....	683
46.3.1	WWDG.....	683
<b>47</b>	<b>FAQs.....</b>	<b>687</b>
<b>48</b>	<b>Revision history .....</b>	<b>691</b>

## List of tables

Table 1: Acronyms and definitions .....	41
Table 2: HAL drivers files .....	43
Table 3: User-application files .....	45
Table 4: APis classification .....	49
Table 5: List of devices supported by the HAL drivers .....	49
Table 6: HAL API naming rules .....	50
Table 7: Macros handling interrupts and specific clock configurations .....	52
Table 8: Callback functions .....	53
Table 9: HAL generic APIs .....	54
Table 10: HAL extension APIs .....	55
Table 11: Define statements used for HAL configuration .....	58
Table 12: Description of GPIO_InitTypeDef structure .....	61
Table 13: Description of EXTI configuration macros .....	63
Table 14: MSP functions .....	67
Table 15: Timeout values .....	71
Table 16: Number of wait states (WS) according to CPU clock (HCLK) frequency .....	418
Table 17: USART frame formats .....	533
Table 18: Document revision history .....	691

## **List of figures**

Figure 1: Example of project template .....	46
Figure 2: Adding family-specific functions .....	55
Figure 3: Adding new peripherals .....	56
Figure 4: Updating existing APIs .....	56
Figure 5: File inclusion model .....	57
Figure 6: HAL driver model .....	65

# 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
CRC	CRC calculation unit
DAC	Digital to analog converter
DCMI	Digital Camera Module Interface
DMA	Direct Memory Access
ETH	Ethernet controller
EXTI	External interrupt/event controller
FLASH	Flash memory
FSMC	Flexible Static 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
MSP	MCU Specific Package
NAND	NAND Flash memory
NOR	Nor Flash memory
NVIC	Nested Vectored Interrupt Controller
PCCARD	PCCARD external memory
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RNG	Random Number Generator
RTC	Real-time clock
SD	Secure Digital

Acronym	Definition
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 a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: USART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode.
- 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>stm32f2xx_hal_ppp.c</i>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32f2xx_hal_adc.c, stm32f2xx_hal_irda.c, ...</i>
<i>stm32f2xx_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: stm32f2xx_hal_adc.h, stm32f2xx_hal_irda.h, ...</i>

File	Description
<i>stm32f2xx_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: stm32f2xx_hal_adc_ex.c, stm32f2xx_hal_dma_ex.c, ...</i>
<i>stm32f2xx_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: stm32f2xx_hal_adc_ex.h, stm32f2xx_hal_dma_ex.h, ...</i>
<i>stm32f2xx_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: stm32f2xx_ll_fsmc.c offers a set of APIs used by stm32f2xx_hal_sram.c, stm32f2xx_hal_nor.c, stm32f2xx_hal_nand.c ...</i>
<i>stm32f2xx_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: stm32f2xx_ll_fsmc.h, stm32f2xx_ll_usb.h, ...</i>
<i>stm32f2xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32f2xx_hal.h</i>	xx_hal.c header file
<i>stm32f2xx_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>stm32f2xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32f2xx_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_stm32f2xx.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 peripheral to use as data memory the external SRAM mounted on the evaluation board.</li> </ul>
<i>startup_stm32f2xx.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>stm32f2xx_flash.icf (optional)</i>	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32f2xx_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>stm32f2xx_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>stm32f2xx_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>stm32f2xx_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:

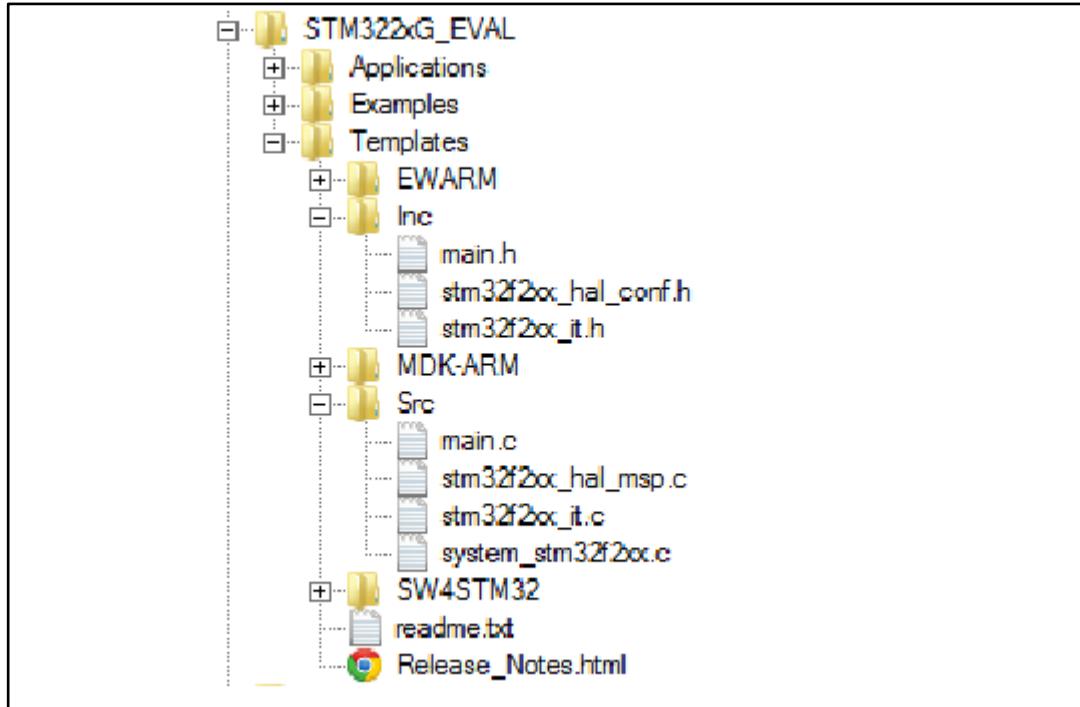
- It contains sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows to configure the CMSIS and HAL drivers accordingly.

- 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 simultaneously.

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

The peripheral handle is used for the following purposes:

- Multi instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.

- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.  
Example: global pointers, DMA handles, state machine.
- Storage : this handle is used also to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
    USART_TypeDef *Instance; /* USART registers base address */
    USART_InitTypeDef Init; /* Usart communication parameters */
    uint8_t *pTxBuffPtr; /* Pointer to Usart Tx transfer Buffer */
    uint16_t TxXferSize; /* Usart Tx Transfer size */
    IO uint16_t TxXferCount; /* Usart Tx Transfer Counter */
    uint8_t *pRxBuffPtr; /* Pointer to Usart Rx transfer Buffer */
    uint16_t RxXferSize; /* Usart Rx Transfer size */
    __IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
    DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
    DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
    HAL_LockTypeDef Lock; /* Locking object */
    __IO HAL_USART_StateTypeDef State; /* Usart communication state */
    __IO uint32_t ErrorCode; /* USART Error code */
}USART_HandleTypeDef;
```



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

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



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



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

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

## 2.2.2 Initialization and configuration structure

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

```
typedef struct
{
    uint32_t BaudRate; /*!< This member configures the UART communication baudrate.*/
    uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received
                          in a frame.*/
    uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
    uint32_t Parity; /*!< Specifies the parity mode. */
    uint32_t Mode; /*!< Specifies 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);
```

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

## 2.4 Devices supported by HAL drivers

**Table 5: List of devices supported by the HAL drivers**

IP/Module	STM32F205xx	STM32F215xx	STM32F207xx	STM32f217xx
stm32F2xx_hal.c	Yes	Yes	Yes	Yes
stm32F2xx_adc.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_adc_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_can.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_cortex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_crc.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_cryp.c	No	Yes	No	Yes
stm32f2xx_hal_dac.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_dac_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_dcmi.c	No	No	Yes	Yes
stm32f2xx_hal_dma.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_dma_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_eth.c	No	No	Yes	Yes
stm32f2xx_hal_flash.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_flash_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_gpio.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_hash.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_hcd.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_i2c.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_i2s.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_irda.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_iwdg.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_nand.c	Yes	Yes	Yes	Yes

IP/Module	STM32F205xx	STM32F215xx	STM32F207xx	STM32f217xx
stm32f2xx_hal_nor.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_pccard.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_pcd.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_pcd_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_pwr.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_rcc.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_rng.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_rtc.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_sd.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_smartcard.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_spi.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_sram.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_tim.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_tim_ex.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_uart.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_usart.c	Yes	Yes	Yes	Yes
stm32f2xx_hal_wwdg.c	Yes	Yes	Yes	Yes
stm32f2xx_ll_fsmc.c	Yes	Yes	Yes	Yes
stm32f2xx_ll_sdmmc.c	Yes	Yes	Yes	Yes
stm32f2xx_ll_usb.c	Yes	Yes	Yes	Yes

## 2.5 HAL drivers rules

### 2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6: HAL API naming rules

	Generic	Family specific	Device specific
File names	<i>stm32f2xx_hal_ppp (c/h)</i>	<i>stm32f2xx_hal_ppp_ex (c/h)</i>	<i>stm32f2xx_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

	Generic	Family specific	Device specific
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 stm32f2xx reference manuals.
- Peripheral registers are declared in the PPP\_TypeDef structure (e.g. ADC\_TypeDef) in stm32f2xxx.h header file. stm32f2xxx.h corresponds to stm32f205xx.h , stm32f215.h , stm32f207xx.h or stm32f217xx.h.
- Peripheral function names are prefixed by HAL\_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. HAL\_UART\_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP\_InitTypeDef (e.g. ADC\_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP\_xxxxConfTypeDef (e.g. ADC\_ChannelConfTypeDef).
- Peripheral handle structures are named PPP\_HandleTypeDef (e.g DMA\_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP\_InitTypeDef are named HAL\_PPP\_Init (e.g. HAL\_TIM\_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named PPP\_DeInit, e.g. TIM\_DeInit.
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL\_PPP\_Function\_DMA ()*.
- The **Feature** prefix should refer to the new feature.  
Example: *HAL\_ADC\_Start()* refers to the injection mode

## 2.5.2 HAL general naming rules

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

Example: The *HAL\_GPIO\_Init()* requires only the GPIO address and its configuration parameters.

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

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

**Table 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 `stm32f2xx_hal_cortex.c` file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example : `STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)"`.
- The PPP handles are valid before using the `HAL_PPP_Init()` API. The init function performs a check before modifying the handle fields.

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

- The macros defined below are used:
  - Conditional macro:

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

- Pseudo-code macro (multiple instructions macro):

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

### 2.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL\_PPP\_IRQHandler() peripheral interrupt handler that should be called from stm32f2xx\_it.c
- User callback functions.

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

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL\_PPP\_MspInit() and HAL\_PPP\_MspDelInit
- Process complete callbacks : HAL\_PPP\_ProcessCpltCallback
- Error callback: HAL\_PPP\_ErrorCallback.

**Table 8: Callback functions**

Callback functions	Example
HAL_PPP_MspInit() / _DelInit()	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\_DelInit()
- **IO operation functions:** HAL\_PPP\_Read(), HAL\_PPP\_Write(), HAL\_PPP\_Transmit(), HAL\_PPP\_Receive()
- **Control functions:** HAL\_PPP\_Set(), HAL\_PPP\_Get().
- **State and Errors functions:** HAL\_PPP\_GetState(), HAL\_PPP\_GetError().

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

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

The initialization and de-initialization functions allow initializing a peripheral and configuring the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The *HAL\_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_DeInit()</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_ConfigChannel()</i>	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	<i>HAL_ADC_AnalogWDGConfig</i>	This function configures the analog watchdog for the selected ADC
<i>State and Errors group</i>	<i>HAL_ADC_GetState()</i>	This function allows getting in runtime the peripheral and the data flow states.
	<i>HAL_ADC_GetError()</i>	This function allows getting in runtime the error that occurred during IT routine

## 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, `stm32f2xx_hal_ppp_ex.c`, that includes all the specific functions and define statements (`stm32f2xx_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)

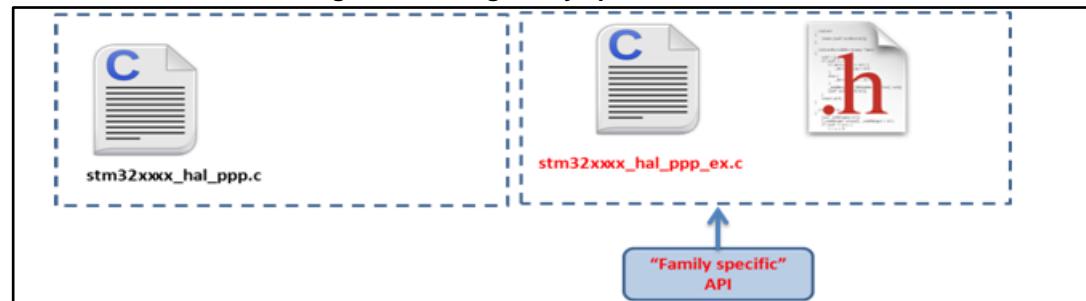
### 2.7.2 HAL extension model cases

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

#### Case 1: Adding a family-specific function

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

Figure 2: Adding family-specific functions



Example: `stm32f2xx_hal_adc_ex.c/h`

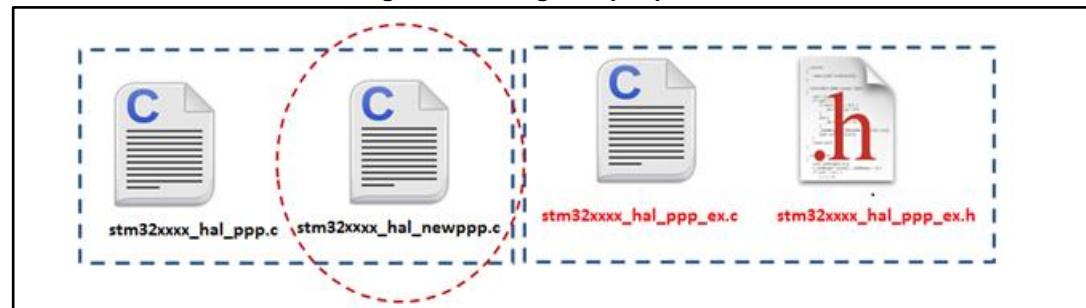
```
HAL_StatusTypeDef HAL_ADCEx_InjectedStop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT(ADC_HandleTypeDef* hadc);
```

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

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32f2xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32lx_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

**Figure 3: Adding new peripherals**

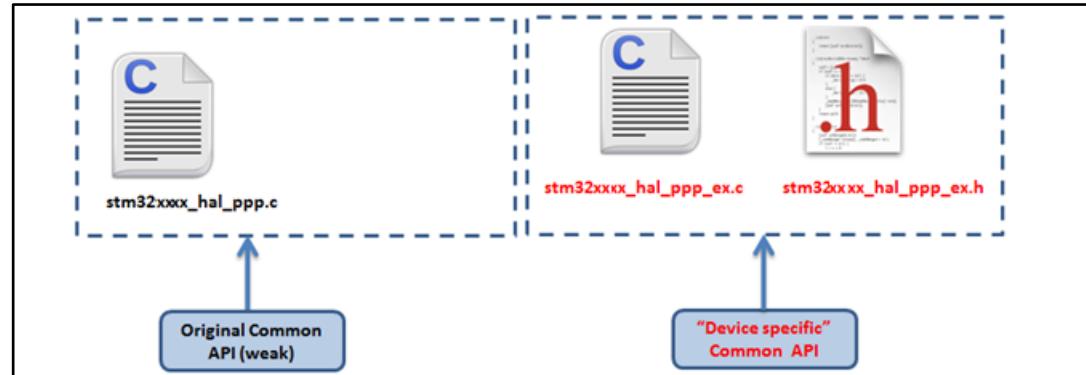


Example: `stm32f2xx_hal_dcmi.c/h`

### Case 3: Updating existing common APIs

In this case, the routines are defined with the same names in the `stm32f2xx_hal_ppp_ex.c` extension file, while the generic API is defined as *weak*, so that the compiler will overwrite the original routine by the new defined function.

**Figure 4: Updating existing APIs**



### Case 4 : 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:

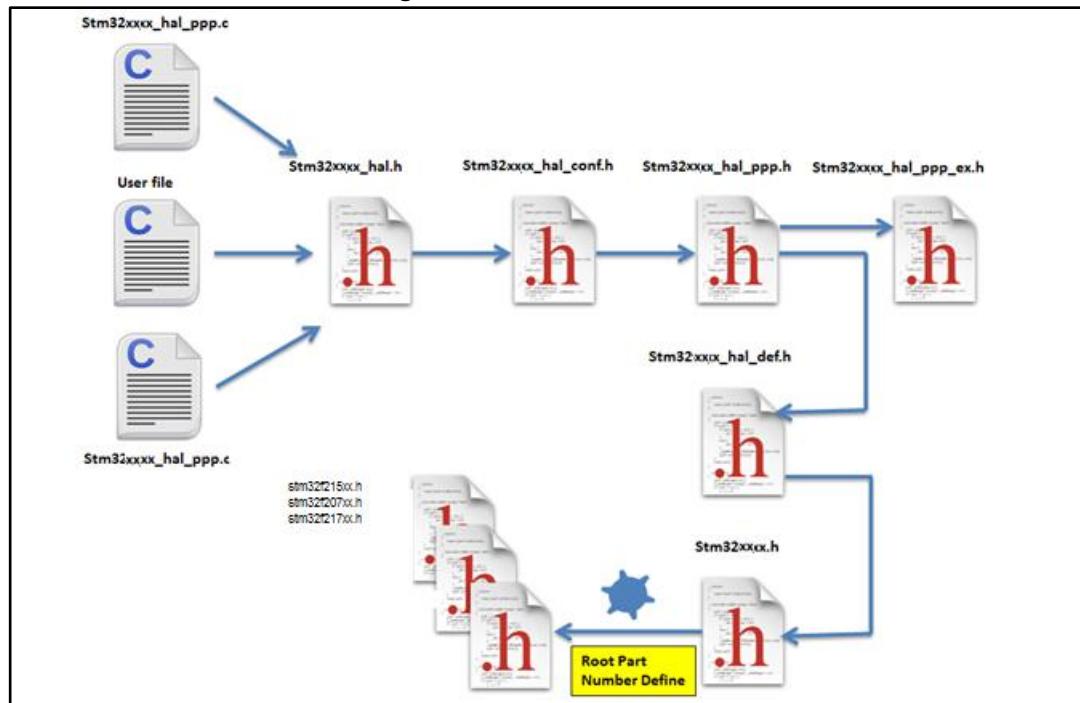
```
#if defined (STM32F205xx)
typedef struct
{
(...)

}PPP_InitTypeDef;
#endif /* STM32F205xx */
```

## 2.8 File inclusion model

The header of the common HAL driver file (stm32f2xx\_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 5: File inclusion model



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE\_HAL\_PPP\_MODULE define statement in the configuration file.

```

/*
 * @file stm32f2xx_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 *stm32f2xx\_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_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 stm32f2xx\_hal\_def.h file calls the stm32f2xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:

- Declarations of peripheral registers and bits definition.
- Macros to access peripheral registers hardware (Write register, Read register...etc.).

- **Common macros**

- Macro defining HAL\_MAX\_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

- Macro linking a PPP peripheral to a DMA structure pointer: \_\_HAL\_LINKDMA();

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

## 2.10 HAL configuration

The configuration file, stm32f2xx\_hal\_conf.h, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

Table 11: Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 (Hz)
HSE_STARTUP_TIMEOUT	Timeout for HSE start up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
EXTERNAL_CLOCK_VALUE	This value is used by the I2S HAL module to compute the I2S clock source frequency, this source is inserted directly through I2S_CKIN pad.	12288000 (Hz)
VDD_VALUE	VDD value	3300 (mV)
USERTOS	Enables the use of RTOS	FALSE (for future use)

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



The `stm32f2xx_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 `stm32f2xx_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, Audio, PLL...). In this case, the clock configuration is performed by an extended API defined in `stm32f2xx_hal_rcc_ex.c`: `HAL_RCCE_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 `stm32f2xx_hal_rcc.h` and `stm32f2xx_hal_rcc_ex.h`. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

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

### 2.11.2 GPIOs

GPIO HAL APIs are the following:

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

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

When selecting EXTI mode with interrupt generation, the user must call `HAL_GPIO_EXTI_IRQHandler()` from `stm32f2xx_it.c` and implement `HAL_GPIO_EXTI_Callback()`

The table below describes the `GPIO_InitTypeDef` structure field.

**Table 12: Description of GPIO\_InitTypeDef structure**

<b>Structure field</b>	<b>Description</b>
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: <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	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_LOW GPIO_SPEED_MEDIUM GPIO_SPEED_HIGH

Please find below typical GPIO configuration examples:

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

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

- Configuring PA0 as external interrupt with falling edge sensitivity:

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

### 2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, `stm32f2xx_hal_cortex.c`, provides APIs to handle NVIC and SysTick. The supported APIs include:

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

### 2.11.4 PWR

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

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

Depending on the STM32 Series, extension functions are available in `stm32f2xx_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 power-down control:
  - `HAL_PWREx_EnableFlashPowerDown()`
  - `HAL_PWREx_DisableFlashPowerDown()`.

### 2.11.5 EXTI

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

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

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

**Table 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>
<code>_HAL_PPP_EXTI_ENABLE_IT(__EXTI_LINE__)</code>	Enables a given EXTI line Example: <code>_HAL_PVD_EXTI_ENABLE_IT(PWR_EXTI_LINE_PVD)</code>
<code>_HAL_PPP_EXTI_DISABLE_IT(__EXTI_LINE__)</code>	Disables a given EXTI line. Example: <code>_HAL_PVD_EXTI_DISABLE_IT(PWR_EXTI_LINE_PVD)</code>
<code>_HAL_PPP_EXTI_GET_FLAG(__EXTI_LINE__)</code>	Gets a given EXTI line interrupt flag pending bit status. Example: <code>_HAL_PVD_EXTI_GET_FLAG(PWR_EXTI_LINE_PVD)</code>
<code>_HAL_PPP_EXTI_CLEAR_FLAG(__EXTI_LINE__)</code>	Clears a given EXTI line interrupt flag pending bit. Example: <code>_HAL_PVD_EXTI_CLEAR_FLAG(PWR_EXTI_LINE_PVD)</code>
<code>_HAL_PPP_EXTI_GENERATE_SWIT(__EXTI_LINE__)</code>	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 `stm32f2xx_it.c` file, and implement `HAL_PPP_FUNCTIONCallback()` callback function (for example `HAL_PWR_PVDCallback()`).

## 2.11.6 DMA

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

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

- Transfer Direction
- Source and Destination data formats
- Circular, Normal or peripheral flow control mode
- 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 Channels.
- \_\_HAL\_DMA\_DISABLE: disables the specified DMA Channels.
- \_\_HAL\_DMA\_GET\_FS: returns the current DMA Stream FIFO filled level.
- \_\_HAL\_DMA\_GET\_FLAG: gets the DMA Channels pending flags.
- \_\_HAL\_DMA\_CLEAR\_FLAG: clears the DMA Channels pending flags.
- \_\_HAL\_DMA\_ENABLE\_IT: enables the specified DMA Channels interrupts.
- \_\_HAL\_DMA\_DISABLE\_IT: disables the specified DMA Channels interrupts.
- \_\_HAL\_DMA\_GET\_IT\_SOURCE: checks whether the specified DMA stream interrupt has occurred or not.



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



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



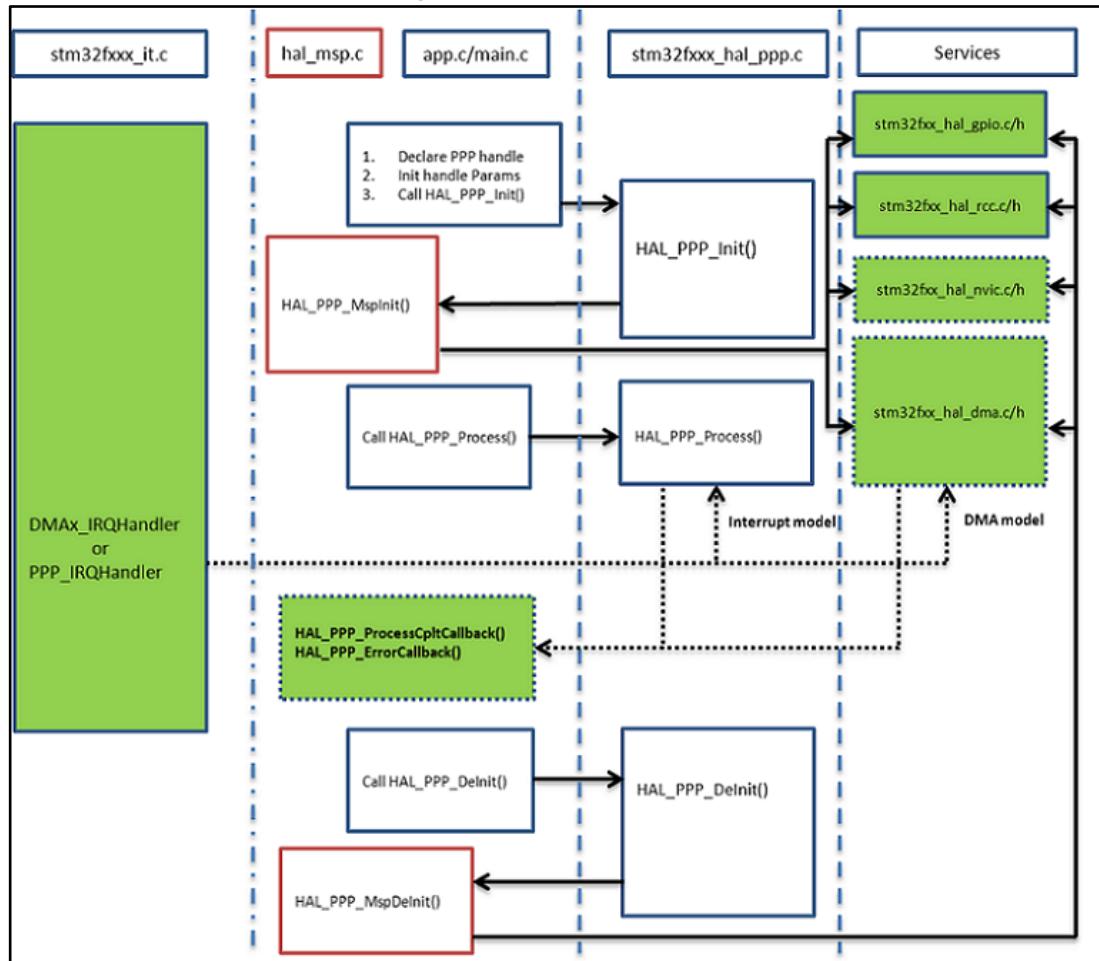
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

## 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 6: HAL driver model



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

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

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

## 2.12.2 HAL initialization

### 2.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file `stm32f2xx_hal.c`.

- `HAL_Init()`: this function must be called at application startup to
  - Initialize data/instruction cache and pre-fetch queue
  - Set Systick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - Call `HAL_MspInit()` user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). `HAL_MspInit()` is defined as “weak” empty function in the HAL drivers.
- `HAL_DeInit()`
  - Resets all peripherals
  - Calls function `HAL_MspDeInit()` which a is user callback function to do system level De-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.

### 2.12.2.2 System clock initialization

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

```
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 = 240;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP DIV2;
RCC_OscInitStruct.PLL.PLLQ = 5;
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_3); }
```

### 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 *stm32f2xx\_hal\_msp.c* file in the user folders. An *stm32f2xx\_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.

*stm32f2xx\_hal\_msp.c* file contains the following functions:

Table 14: MSP functions

Routine	Description
<b>void HAL_MspInit()</b>	Global MSP initialization routine
<b>void HAL_MspDeInit()</b>	Global MSP de-initialization routine
<b>void HAL_PPP_MspInit()</b>	PPP MSP initialization routine
<b>void HAL_PPP_MspDeInit()</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\_MspDeInit()*. In this case the *HAL\_PPP\_MspInit()* and *HAL\_PPP\_MspDeInit()* are not implemented.

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

If there is nothing to be initialized by the global *HAL\_MspInit()* and *HAL\_MspDeInit()*, the two routines can simply be omitted.

### 2.12.3 HAL IO operation process

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

- Polling mode
- Interrupt mode
- DMA mode

#### 2.12.3.1 Polling mode

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

The example below shows the typical polling mode processing sequence :

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

#### 2.12.3.2 Interrupt mode

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

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

- *HAL\_PPP\_Process\_IT()*: launch the process
- *HAL\_PPP\_IRQHandler()*: the global PPP peripheral interruption
- *\_\_weak HAL\_PPP\_ProcessCpltCallback ()*: the callback relative to the process completion.
- *\_\_weak HAL\_PPP\_ProcessErrorCallback()*: the callback relative to the process Error.

To use a process in interrupt mode, *HAL\_PPP\_Process\_IT()* is called in the user file and *HAL\_PPP\_IRQHandler* in *stm32f2xx\_it.c*.

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

An example of use is illustrated below:

*main.c* file:

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

*stm32f2xx\_it.c* file:

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

### 2.12.3.3 DMA mode

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

- *HAL\_PPP\_Process\_DMA()*: launch the process
- *HAL\_PPP\_DMA\_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *\_\_weak HAL\_PPP\_ProcessCpltCallback()*: the callback relative to the process completion.
- *\_\_weak HAL\_PPP\_ErrorCpltCallback()*: the callback relative to the process Error.

To use a process in DMA mode, *HAL\_PPP\_Process\_DMA()* is called in the user file and the *HAL\_PPP\_DMA\_IRQHandler()* is placed in the *stm32f2xx\_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 = USART1;
HAL_UART_Init(&UartHandle);
(...)

void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(...)
__HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
    HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(...)}
}

```

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

An example of use is illustrated below:

*main.c* file:

```

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

```

*stm32f2xx\_it.c* file:

```

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

```

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

```

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

```

## 2.12.4 Timeout and error management

### 2.12.4.1 Timeout management

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

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

The timeout possible value are the following:

**Table 15: Timeout values**

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

**Notes:**

<sup>(1)</sup>HAL\_MAX\_DELAY is defined in the stm32f2xx\_hal\_def.h as 0xFFFFFFFF

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

Example of fixed timeout:

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

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

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

```

    _HAL_UNLOCK(hppp);
    hppp->State= HAL_PPP_STATE_TIMEOUT;
    return hppp->State;
}
}
(...)
```

#### 2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

```

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

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

```

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

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

```

{
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_HandleTypeDef State; /* PPP state */
    IO_HandleTypeDef 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 *stm32f2xx\_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 *stm32f2xx\_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 System Driver

### 3.1 HAL Firmware driver API description

#### 3.1.1 How to use this driver

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

The HAL contains two APIs categories:

- Common HAL APIs
- Services HAL APIs

#### 3.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [\*\*HAL\\_Init\(\)\*\*](#)
- [\*\*HAL\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_MspDeInit\(\)\*\*](#)
- [\*\*HAL\\_InitTick\(\)\*\*](#)

#### 3.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier

- Get the device revision identifier
- Enable/Disable Debug module during SLEEP mode
- Enable/Disable Debug module during STOP mode
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:

- [\*HAL\\_IncTick\(\)\*](#)
- [\*HAL\\_GetTick\(\)\*](#)
- [\*HAL\\_Delay\(\)\*](#)
- [\*HAL\\_SuspendTick\(\)\*](#)
- [\*HAL\\_ResumeTick\(\)\*](#)
- [\*HAL\\_GetHalVersion\(\)\*](#)
- [\*HAL\\_GetREVID\(\)\*](#)
- [\*HAL\\_GetDEVID\(\)\*](#)
- [\*HAL\\_DBGMCU\\_EnableDBGSleepMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_DisableDBGSleepMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_EnableDBGStopMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_DisableDBGStopMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_EnableDBGStandbyMode\(\)\*](#)
- [\*HAL\\_DBGMCU\\_DisableDBGStandbyMode\(\)\*](#)
- [\*HAL\\_EnableCompensationCell\(\)\*](#)
- [\*HAL\\_DisableCompensationCell\(\)\*](#)

### 3.1.4 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.
Return values	<ul style="list-style-type: none"><li>• HAL status</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>

### 3.1.5 HAL\_DelInit

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

### 3.1.6 HAL\_MspInit

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

### 3.1.7 HAL\_MspDelInit

Function Name	<b>void HAL_MspDeInit (void )</b>
Function Description	Deinitializes the MSP.
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
<b>3.1.8 HAL_InitTick</b>	
Function Name	<b>HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)</b>
Function Description	This function configures the source of the time base.
Parameters	<ul style="list-style-type: none"> <li><b>TickPriority:</b> Tick interrupt priority.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig().</li> <li>In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.</li> <li>The function is declared as __weak to be overwritten in case of other implementation in user file.</li> </ul>
<b>3.1.9 HAL_IncTick</b>	
Function Name	<b>void HAL_IncTick (void )</b>
Function Description	This function is called to increment a global variable "uwTick" used as application time base.
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>In the default implementation, this variable is incremented each 1ms in Systick ISR.</li> <li>This function is declared as __weak to be overwritten in case of other implementations in user file.</li> </ul>
<b>3.1.10 HAL_GetTick</b>	
Function Name	<b>uint32_t HAL_GetTick (void )</b>
Function Description	Provides a tick value in millisecond.
Return values	<ul style="list-style-type: none"> <li>tick value</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function is declared as __weak to be overwritten in case of other implementations in user file.</li> </ul>
<b>3.1.11 HAL_Delay</b>	
Function Name	<b>void HAL_Delay (__IO uint32_t Delay)</b>
Function Description	This function provides accurate delay (in milliseconds) based on variable incremented.
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>In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.</li> <li>This function is declared as __weak to be overwritten in case of other implementations in user file.</li> </ul>

### 3.1.12 HAL\_SuspendTick

Function Name	<b>void HAL_SuspendTick (void )</b>
Function Description	Suspend Tick increment.
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the SysTick interrupt will be disabled and so Tick increment is suspended.</li> <li>This function is declared as __weak to be overwritten in case of other implementations in user file.</li> </ul>

### 3.1.13 HAL\_ResumeTick

Function Name	<b>void HAL_ResumeTick (void )</b>
Function Description	Resume Tick increment.
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the SysTick interrupt will be enabled and so Tick increment is resumed.</li> <li>This function is declared as __weak to be overwritten in case of other implementations in user file.</li> </ul>

### 3.1.14 HAL\_GetHalVersion

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

### 3.1.15 HAL\_GetREVID

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

### 3.1.16 HAL\_GetDEVID

Function Name	<b>uint32_t HAL_GetDEVID (void )</b>
Function Description	Returns the device identifier.

	Return values	• Device identifier
<b>3.1.17</b>	<b>HAL_DBGMCU_EnableDBGSleepMode</b>	
Function Name	<b>void HAL_DBGMCU_EnableDBGSleepMode (void )</b>	
Function Description	Enable the Debug Module during SLEEP mode.	
Return values	• None	
<b>3.1.18</b>	<b>HAL_DBGMCU_DisableDBGSleepMode</b>	
Function Name	<b>void HAL_DBGMCU_DisableDBGSleepMode (void )</b>	
Function Description	Disable the Debug Module during SLEEP mode.	
Return values	• None	
<b>3.1.19</b>	<b>HAL_DBGMCU_EnableDBGStopMode</b>	
Function Name	<b>void HAL_DBGMCU_EnableDBGStopMode (void )</b>	
Function Description	Enable the Debug Module during STOP mode.	
Return values	• None	
<b>3.1.20</b>	<b>HAL_DBGMCU_DisableDBGStopMode</b>	
Function Name	<b>void HAL_DBGMCU_DisableDBGStopMode (void )</b>	
Function Description	Disable the Debug Module during STOP mode.	
Return values	• None	
<b>3.1.21</b>	<b>HAL_DBGMCU_EnableDBGStandbyMode</b>	
Function Name	<b>void HAL_DBGMCU_EnableDBGStandbyMode (void )</b>	
Function Description	Enable the Debug Module during STANDBY mode.	
Return values	• None	
<b>3.1.22</b>	<b>HAL_DBGMCU_DisableDBGStandbyMode</b>	
Function Name	<b>void HAL_DBGMCU_DisableDBGStandbyMode (void )</b>	
Function Description	Disable the Debug Module during STANDBY mode.	
Return values	• None	
<b>3.1.23</b>	<b>HAL_EnableCompensationCell</b>	
Function Name	<b>void HAL_EnableCompensationCell (void )</b>	
Function Description	Enables the I/O Compensation Cell.	
Return values	• None	
Notes	• The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.	

### 3.1.24 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	<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>

## 3.2 HAL Firmware driver defines

### 3.2.1 HAL

#### *HAL CAN Error Code*

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

#### *HAL Exported Macros*

`_HAL_DBGMCU_FREEZE_TIM2`  
`_HAL_DBGMCU_FREEZE_TIM3`  
`_HAL_DBGMCU_FREEZE_TIM4`  
`_HAL_DBGMCU_FREEZE_TIM5`  
`_HAL_DBGMCU_FREEZE_TIM6`  
`_HAL_DBGMCU_FREEZE_TIM7`  
`_HAL_DBGMCU_FREEZE_TIM12`  
`_HAL_DBGMCU_FREEZE_TIM13`  
`_HAL_DBGMCU_FREEZE_TIM14`  
`_HAL_DBGMCU_FREEZE_RTC`  
`_HAL_DBGMCU_FREEZE_WWDG`  
`_HAL_DBGMCU_FREEZE_IWDG`  
`_HAL_DBGMCU_FREEZE_I2C1_TIMEOUT`  
`_HAL_DBGMCU_FREEZE_I2C2_TIMEOUT`  
`_HAL_DBGMCU_FREEZE_I2C3_TIMEOUT`

```
_HAL_DBGMCU_FREEZE_CAN1  
_HAL_DBGMCU_FREEZE_CAN2  
_HAL_DBGMCU_FREEZE_TIM1  
_HAL_DBGMCU_FREEZE_TIM8  
_HAL_DBGMCU_FREEZE_TIM9  
_HAL_DBGMCU_FREEZE_TIM10  
_HAL_DBGMCU_FREEZE_TIM11  
_HAL_DBGMCU_UNFREEZE_TIM2  
_HAL_DBGMCU_UNFREEZE_TIM3  
_HAL_DBGMCU_UNFREEZE_TIM4  
_HAL_DBGMCU_UNFREEZE_TIM5  
_HAL_DBGMCU_UNFREEZE_TIM6  
_HAL_DBGMCU_UNFREEZE_TIM7  
_HAL_DBGMCU_UNFREEZE_TIM12  
_HAL_DBGMCU_UNFREEZE_TIM13  
_HAL_DBGMCU_UNFREEZE_TIM14  
_HAL_DBGMCU_UNFREEZE_RTC  
_HAL_DBGMCU_UNFREEZE_WWDG  
_HAL_DBGMCU_UNFREEZE_IWDG  
_HAL_DBGMCU_UNFREEZE_I2C1_TIMEOUT  
_HAL_DBGMCU_UNFREEZE_I2C2_TIMEOUT  
_HAL_DBGMCU_UNFREEZE_I2C3_TIMEOUT  
_HAL_DBGMCU_UNFREEZE_CAN1  
_HAL_DBGMCU_UNFREEZE_CAN2  
_HAL_DBGMCU_UNFREEZE_TIM1  
_HAL_DBGMCU_UNFREEZE_TIM8  
_HAL_DBGMCU_UNFREEZE_TIM9  
_HAL_DBGMCU_UNFREEZE_TIM10  
_HAL_DBGMCU_UNFREEZE_TIM11  
_HAL_SYSCFG_REMAPMEMORY_FLASH  
_HAL_SYSCFG_REMAPMEMORY_SYSTEMFLASH  
_HAL_SYSCFG_REMAPMEMORY_SRAM  
_HAL_SYSCFG_REMAPMEMORY_FSMC
```

**HAL Private Constants**

```
_STM32F2xx_HAL_VERSION_MAIN [31:24] main version  
_STM32F2xx_HAL_VERSION_SUB1 [23:16] sub1 version
```

---

\_\_STM32F2xx\_HAL\_VERSION\_SUB2 [15:8] sub2 version  
\_\_STM32F2xx\_HAL\_VERSION\_RC [7:0] release candidate  
\_\_STM32F2xx\_HAL\_VERSION  
IDCODE\_DEVID\_MASK  
SYSCFG\_OFFSET  
MEMRMP\_OFFSET  
UFB\_MODE\_BIT\_NUMBER  
UFB\_MODE\_BB  
CMPCR\_OFFSET  
CMP\_PD\_BIT\_NUMBER  
CMPCR\_CMP\_PD\_BB

## 4 HAL ADC Generic Driver

### 4.1 ADC Firmware driver registers structures

#### 4.1.1 ADC\_InitTypeDef

##### Data Fields

- *uint32\_t ClockPrescaler*
- *uint32\_t Resolution*
- *uint32\_t DataAlign*
- *uint32\_t ScanConvMode*
- *uint32\_t EOCSelection*
- *uint32\_t ContinuousConvMode*
- *uint32\_t NbrOfConversion*
- *uint32\_t DiscontinuousConvMode*
- *uint32\_t NbrOfDiscConversion*
- *uint32\_t ExternalTrigConv*
- *uint32\_t ExternalTrigConvEdge*
- *uint32\_t DMAContinuousRequests*

##### Field Documentation

- ***uint32\_t ADC\_InitTypeDef::ClockPrescaler***  
Select ADC clock prescaler. 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. This parameter can be a value of [\*\*ADC\\_Resolution\*\*](#)
- ***uint32\_t ADC\_InitTypeDef::DataAlign***  
Specifies ADC data alignment to right (MSB on register bit 11 and LSB on register bit 0) (default setting) or to left (if regular group: MSB on register bit 15 and LSB on register bit 4, if injected group (MSB kept as signed value due to potential negative value after offset application): MSB on register bit 14 and LSB on register bit 3). This parameter can be a value of [\*\*ADC\\_Data\\_align\*\*](#)
- ***uint32\_t ADC\_InitTypeDef::ScanConvMode***  
Configures the sequencer of regular and injected groups. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion'/'InjectedNbrOfConversion' and each channel rank). Scan direction is upward: from rank1 to rank 'n'. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t ADC\_InitTypeDef::EOCSelection***  
Specifies what EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of conversion of each rank or complete sequence. This parameter can be a value of [\*\*ADC\\_EOCSelection\*\*](#). Note: For injected group, end of conversion (flag&IT) is raised only at the end of the sequence. Therefore, if end of conversion is set to end of each conversion, injected group should not be used with interruption (HAL\_ADCEx\_InjectedStart\_IT) or polling (HAL\_ADCEx\_InjectedStart and

`HAL_ADCEx_InjectedPollForConversion)`. By the way, polling is still possible since driver will use an estimated timing for end of injected conversion. Note: If overrun feature is intended to be used, use ADC in mode 'interruption' (function

`HAL_ADC_Start_IT()`) with parameter `EOCSelection` set to end of each conversion or in mode 'transfer by DMA' (function `HAL_ADC_Start_DMA()`). If overrun feature is intended to be bypassed, use ADC in mode 'polling' or 'interruption' with parameter `EOCSelection` must be set to end of sequence

- **`uint32_t ADC_InitTypeDef::ContinuousConvMode`**  
Specifies whether the conversion is performed in single mode (one conversion) or continuous mode for regular group, after the selected trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- **`uint32_t ADC_InitTypeDef::NbrOfConversion`**  
Specifies the number of ranks that will be converted within the regular group sequencer. To use regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min\_Data = 1 and Max\_Data = 16.
- **`uint32_t ADC_InitTypeDef::DiscontinuousConvMode`**  
Specifies whether the conversions sequence of regular group is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.
- **`uint32_t ADC_InitTypeDef::NbrOfDiscConversion`**  
Specifies the number of discontinuous conversions in which the main sequence of regular group (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min\_Data = 1 and Max\_Data = 8.
- **`uint32_t ADC_InitTypeDef::ExternalTrigConv`**  
Selects the external event used to trigger the conversion start of regular group. If set to `ADC_SOFTWARE_START`, external triggers are disabled. If set to external trigger source, triggering is on event rising edge by default. This parameter can be a value of [`ADC\_External\_trigger\_Source-Regular`](#)
- **`uint32_t ADC_InitTypeDef::ExternalTrigConvEdge`**  
Selects the external trigger edge of regular group. If trigger is set to `ADC_SOFTWARE_START`, this parameter is discarded. This parameter can be a value of [`ADC\_External\_trigger\_edge-Regular`](#)
- **`uint32_t ADC_InitTypeDef::DMAContinuousRequests`**  
Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion). This parameter can be set to ENABLE or DISABLE.

#### 4.1.2 `ADC_ChannelConfTypeDef`

##### Data Fields

- **`uint32_t Channel`**

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

#### Field Documentation

- *uint32\_t ADC\_ChannelConfTypeDef::Channel*  
Specifies the channel to configure into ADC regular group. This parameter can be a value of [ADC\\_channels](#)
- *uint32\_t ADC\_ChannelConfTypeDef::Rank*  
Specifies 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*  
Sampling time value to be set for the selected channel. Unit: ADC clock cycles  
Conversion time is the addition of sampling time and processing time (12 ADC clock cycles at ADC resolution 12 bits, 11 cycles at 10 bits, 9 cycles at 8 bits, 7 cycles at 6 bits). This parameter can be a value of [ADC\\_sampling\\_times](#) Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS\_vrefint, TS\_temp (values rough order: 4us min).
- *uint32\_t ADC\_ChannelConfTypeDef::Offset*  
Reserved for future use, can be set to 0

### 4.1.3 ADC\_AnalogWDGConfTypeDef

#### 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

#### 4.1.4 ADC\_HandleTypeDef

##### Data Fields

- ***ADC\_TypeDef \* Instance***
- ***ADC\_InitTypeDef Init***
- ***\_IO uint32\_t NbrOfCurrentConversionRank***
- ***DMA\_HandleTypeDef \* DMA\_Handle***
- ***HAL\_LockTypeDef Lock***
- ***\_IO uint32\_t 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 uint32\_t ADC\_HandleTypeDef::State***  
ADC communication state
- ***\_IO uint32\_t ADC\_HandleTypeDef::ErrorCode***  
ADC Error code

## 4.2 ADC Firmware driver API description

### 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:  $V_{REF(minus)} = V_{IN} = V_{REF(plus)}$ .
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 `__HAL_RCC_ADC_CLK_ENABLE()`
  - b. ADC pins configuration
    - Enable the clock for the ADC GPIOs using the following function:  
`__HAL_RCC_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  
`__HAL_RCC_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.

#### Configuration of ADC, groups regular/injected, channels parameters

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

#### Execution of ADC conversions

1. ADC driver can be used among three modes: polling, interruption, transfer by DMA.

### 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
- ADC\_GET\_RESOLUTION: Return resolution bits in CR1 register



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

## Deinitialization of ADC

1. Disable the ADC interface
  - ADC clock can be hard reset and disabled at RCC top level.
  - Hard reset of ADC peripherals using macro  
  \_\_HAL\_RCC\_ADC\_FORCE\_RESET(), \_\_HAL\_RCC\_ADC\_RELEASE\_RESET().
  - ADC clock disable using the equivalent macro/functions as configuration step.
    - Example: Into HAL\_ADC\_MspDelinit() (recommended code location) or with other device clock parameters configuration:
    - HAL\_RCC\_GetOscConfig(&RCC\_OsclInitStructure);
    - RCC\_OsclInitStructure.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;
    - RCC\_OsclInitStructure.HSISState = RCC\_HSI\_OFF; (if not used for system clock)
    - HAL\_RCC\_OscConfig(&RCC\_OsclInitStructure);
2. ADC pins configuration
  - Disable the clock for the ADC GPIOs using macro  
  \_\_HAL\_RCC\_GPIOx\_CLK\_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
  - Disable the NVIC for ADC using function HAL\_NVIC\_DisableIRQ(ADCx\_IRQn)
4. Optionally, in case of usage of DMA:
  - Deinitialize the DMA using function HAL\_DMA\_Delinit().
  - Disable the NVIC for DMA using function  
  HAL\_NVIC\_DisableIRQ(DMAx\_Channelx\_IRQn)

### 4.2.3

## Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- De-initialize the ADC.

This section contains the following APIs:

- [\*\*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.

This section contains the following APIs:

- [\*\*HAL\\_ADC\\_Start\(\)\*\*](#)
- [\*\*HAL\\_ADC\\_Stop\(\)\*\*](#)
- [\*\*HAL\\_ADC\\_PollForConversion\(\)\*\*](#)
- [\*\*HAL\\_ADC\\_PollForEvent\(\)\*\*](#)

- [\*HAL\\_ADC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_ADC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_ADC\\_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.

This section contains the following APIs:

- [\*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

This section contains the following APIs:

- [\*HAL\\_ADC\\_GetState\(\)\*](#)
- [\*HAL\\_ADC\\_GetError\(\)\*](#)

#### 4.2.7 HAL\_ADC\_Init

Function Name	<code>HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef *hadc)</code>
Function Description	Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct and 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>• HAL status</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>

#### 4.2.8 HAL\_ADC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_ADC_DelInit (ADC_HandleTypeDef * 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 ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 4.2.9 HAL\_ADC\_MspInit

Function Name	<b>void HAL_ADC_MspInit (ADC_HandleTypeDef * 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>

#### 4.2.10 HAL\_ADC\_MspDelInit

Function Name	<b>void HAL_ADC_MspDelInit (ADC_HandleTypeDef * 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>

#### 4.2.11 HAL\_ADC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * 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>• HAL status</li> </ul>

#### 4.2.12 HAL\_ADC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * 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>• HAL status.</li> </ul>

Notes

- Caution: This function will stop also injected channels.

#### 4.2.13 HAL\_ADC\_PollForConversion

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * 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>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function.</li> <li>This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence.</li> </ul>

#### 4.2.14 HAL\_ADC\_PollForEvent

Function Name	<b>HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * 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: ADC_AWD_EVENT: ADC Analog watch Dog event. ADC_OVR_EVENT: ADC Overrun event.</li> <li><b>Timeout:</b> Timeout value in millisecond.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 4.2.15 HAL\_ADC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)</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>HAL status.</li> </ul>

#### 4.2.16 HAL\_ADC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)</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</li> </ul>

contains the configuration information for the specified ADC.

- |               |  |
|---------------|--|
| Return values | <ul style="list-style-type: none"> <li>• HAL status.</li> </ul>  |
| Notes         | <ul style="list-style-type: none"> <li>• Caution: This function will stop also injected channels.</li> </ul> |

#### 4.2.17 HAL\_ADC\_IRQHandler

- |                      |  |
|----------------------|--|
| Function Name        | <b>void HAL_ADC_IRQHandler (ADC_HandleTypeDef * 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>   |

#### 4.2.18 HAL\_ADC\_Start\_DMA

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * 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>• HAL status</li> </ul>  |

#### 4.2.19 HAL\_ADC\_Stop\_DMA

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * 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 contains the configuration information for the specified ADC.</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |

#### 4.2.20 HAL\_ADC\_GetValue

- |                      |  |
|----------------------|--|
| Function Name        | <b>uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)</b>  |
| 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 ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• Converted value</li> </ul>  |

#### 4.2.21 HAL\_ADC\_ConvCpltCallback

Function Name	<b>void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)</b>
Function Description	Regular conversion complete 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>

#### 4.2.22 HAL\_ADC\_ConvHalfCpltCallback

Function Name	<b>void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)</b>
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>

#### 4.2.23 HAL\_ADC\_LevelOutOfWindowCallback

Function Name	<b>void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * 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>

#### 4.2.24 HAL\_ADC\_ErrorCallback

Function Name	<b>void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * 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>

#### 4.2.25 HAL\_ADC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * 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>• HAL status</li> </ul>

#### 4.2.26 HAL\_ADC\_AnalogWDGConfig

Function Name	<code>HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)</code>
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>• HAL status</li> </ul>

#### 4.2.27 HAL\_ADC\_GetState

Function Name	<code>uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)</code>
Function Description	return the ADC state
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>• HAL state</li> </ul>

#### 4.2.28 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>• ADC Error Code</li> </ul>

### 4.3 ADC Firmware driver defines

#### 4.3.1 ADC

##### *ADC Analog Watchdog Selection*

`ADC_ANALOGWATCHDOG_SINGLE_REG`  
`ADC_ANALOGWATCHDOG_SINGLE_INJEC`  
`ADC_ANALOGWATCHDOG_SINGLE_REGINJEC`  
`ADC_ANALOGWATCHDOG_ALL_REG`  
`ADC_ANALOGWATCHDOG_ALL_INJEC`  
`ADC_ANALOGWATCHDOG_ALL_REGINJEC`  
`ADC_ANALOGWATCHDOG_NONE`

##### *ADC Common Channels*

`ADC_CHANNEL_0`  
`ADC_CHANNEL_1`

ADC\_CHANNEL\_2  
ADC\_CHANNEL\_3  
ADC\_CHANNEL\_4  
ADC\_CHANNEL\_5  
ADC\_CHANNEL\_6  
ADC\_CHANNEL\_7  
ADC\_CHANNEL\_8  
ADC\_CHANNEL\_9  
ADC\_CHANNEL\_10  
ADC\_CHANNEL\_11  
ADC\_CHANNEL\_12  
ADC\_CHANNEL\_13  
ADC\_CHANNEL\_14  
ADC\_CHANNEL\_15  
ADC\_CHANNEL\_16  
ADC\_CHANNEL\_17  
ADC\_CHANNEL\_18  
ADC\_CHANNEL\_VREFINT  
ADC\_CHANNEL\_VBAT

***ADC Channels Type***

ADC\_ALL\_CHANNELS  
ADC\_REGULAR\_CHANNELS reserved for future use  
ADC\_INJECTED\_CHANNELS reserved for future use

***ADC Clock Prescaler***

ADC\_CLOCK\_SYNC\_PCLK\_DIV2  
ADC\_CLOCK\_SYNC\_PCLK\_DIV4  
ADC\_CLOCK\_SYNC\_PCLK\_DIV6  
ADC\_CLOCK\_SYNC\_PCLK\_DIV8

***ADC Data Align***

ADC\_DATAALIGN\_RIGHT  
ADC\_DATAALIGN\_LEFT

***ADC Delay Between 2 Sampling Phases***

ADC\_TWOSAMPLINGDELAY\_5CYCLES  
ADC\_TWOSAMPLINGDELAY\_6CYCLES  
ADC\_TWOSAMPLINGDELAY\_7CYCLES  
ADC\_TWOSAMPLINGDELAY\_8CYCLES

ADC\_TWOSAMPLINGDELAY\_9CYCLES  
ADC\_TWOSAMPLINGDELAY\_10CYCLES  
ADC\_TWOSAMPLINGDELAY\_11CYCLES  
ADC\_TWOSAMPLINGDELAY\_12CYCLES  
ADC\_TWOSAMPLINGDELAY\_13CYCLES  
ADC\_TWOSAMPLINGDELAY\_14CYCLES  
ADC\_TWOSAMPLINGDELAY\_15CYCLES  
ADC\_TWOSAMPLINGDELAY\_16CYCLES  
ADC\_TWOSAMPLINGDELAY\_17CYCLES  
ADC\_TWOSAMPLINGDELAY\_18CYCLES  
ADC\_TWOSAMPLINGDELAY\_19CYCLES  
ADC\_TWOSAMPLINGDELAY\_20CYCLES

***ADC EOC Selection***

ADC\_EOC\_SEQ\_CONV  
ADC\_EOC\_SINGLE\_CONV  
ADC\_EOC\_SINGLE\_SEQ\_CONV reserved for future use

***ADC Error Code***

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

***ADC Event Type***

ADC\_AWD\_EVENT  
ADC\_OVR\_EVENT

***ADC Exported Macros***

`_HAL_ADC_RESET_HANDLE_STATE` **Description:**

- Reset ADC handle state.

**Parameters:**

- `_HANDLE_`: ADC handle

**Return value:**

- None

`_HAL_ADC_ENABLE`

**Description:**

- Enable the ADC peripheral.

**Parameters:**

- `_HANDLE_`: ADC handle

`__HAL_ADC_DISABLE`

**Return value:**

- None

**Description:**

- Disable the ADC peripheral.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

`__HAL_ADC_ENABLE_IT`

**Description:**

- Enable the ADC end of conversion interrupt.

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__INTERRUPT__`: ADC Interrupt.

**Return value:**

- None

`__HAL_ADC_DISABLE_IT`

**Description:**

- Disable the ADC end of conversion interrupt.

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__INTERRUPT__`: ADC interrupt.

**Return value:**

- None

`__HAL_ADC_GET_IT_SOURCE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__INTERRUPT__`: specifies the ADC interrupt source to check.

**Return value:**

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

`__HAL_ADC_CLEAR_FLAG`

**Description:**

- Clear the ADC's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the ADC Handle.
- `__FLAG__`: ADC flag.

**\_HAL\_ADC\_GET\_FLAG****Return value:**

- None

**Description:**

- Get the selected ADC's flag status.

**Parameters:**

- **\_HANDLE**: specifies the ADC Handle.
- **\_FLAG**: ADC flag.

**Return value:**

- None

**ADC Exported Types**

HAL_ADC_STATE_RESET	ADC not yet initialized or disabled
HAL_ADC_STATE_READY	ADC peripheral ready for use
HAL_ADC_STATE_BUSY_INTERNAL	ADC is busy to internal process (initialization, calibration)
HAL_ADC_STATE_TIMEOUT	TimeOut occurrence
HAL_ADC_STATE_ERROR_INTERNAL	Internal error occurrence
HAL_ADC_STATE_ERROR_CONFIG	Configuration error occurrence
HAL_ADC_STATE_ERROR_DMA	DMA error occurrence
HAL_ADC_STATE_REG_BUSY	A conversion on group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))
HAL_ADC_STATE_REG_EOC	Conversion data available on group regular
HAL_ADC_STATE_REG_OVR	Overrun occurrence
HAL_ADC_STATE_INJ_BUSY	A conversion on group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))
HAL_ADC_STATE_INJ_EOC	Conversion data available on group injected
HAL_ADC_STATE_AWD1	Out-of-window occurrence of analog watchdog 1

**ADC External Trigger Edge Regular**

ADC\_EXTERNALTRIGCONVEDGE\_NONE  
 ADC\_EXTERNALTRIGCONVEDGE\_RISING  
 ADC\_EXTERNALTRIGCONVEDGE\_FALLING  
 ADC\_EXTERNALTRIGCONVEDGE\_RISINGFALLING

**ADC External Trigger Source Regular**

ADC\_EXTERNALTRIGCONV\_T1\_CC1

ADC\_EXTERNALTRIGCONV\_T1\_CC2  
ADC\_EXTERNALTRIGCONV\_T1\_CC3  
ADC\_EXTERNALTRIGCONV\_T2\_CC2  
ADC\_EXTERNALTRIGCONV\_T2\_CC3  
ADC\_EXTERNALTRIGCONV\_T2\_CC4  
ADC\_EXTERNALTRIGCONV\_T2\_TRGO  
ADC\_EXTERNALTRIGCONV\_T3\_CC1  
ADC\_EXTERNALTRIGCONV\_T3\_TRGO  
ADC\_EXTERNALTRIGCONV\_T4\_CC4  
ADC\_EXTERNALTRIGCONV\_T5\_CC1  
ADC\_EXTERNALTRIGCONV\_T5\_CC2  
ADC\_EXTERNALTRIGCONV\_T5\_CC3  
ADC\_EXTERNALTRIGCONV\_T8\_CC1  
ADC\_EXTERNALTRIGCONV\_T8\_TRGO  
ADC\_EXTERNALTRIGCONV\_Ext\_IT11  
ADC\_SOFTWARE\_START

***ADC Flags Definition***

ADC\_FLAG\_AWD  
ADC\_FLAG\_EOC  
ADC\_FLAG\_JEOC  
ADC\_FLAG\_JSTRT  
ADC\_FLAG\_STRT  
ADC\_FLAG\_OVR

***ADC Interrupts Definition***

ADC\_IT\_EOC  
ADC\_IT\_AWD  
ADC\_IT\_JEOC  
ADC\_IT\_OVR

***ADC Private Constants***

ADC\_STAB\_DELAY\_US  
ADC\_TEMPSENSOR\_DELAY\_US

***ADC Private Macros***

ADC\_IS\_ENABLE

**Description:**

- Verification of ADC state: enabled or disabled.

**Parameters:**

- \_\_HANDLE\_\_: ADC handle

**Return value:**

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

ADC\_IS\_SOFTWARE\_START\_REGULAR

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: ADC handle

**Return value:**

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

ADC\_IS\_SOFTWARE\_START\_INJECTED

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: ADC handle

**Return value:**

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

ADC\_STATE\_CLR\_SET

**Description:**

- Simultaneously clears and sets specific bits of the handle State.

**Return value:**

- None

**Notes:**

- : ADC\_STATE\_CLR\_SET() macro is merely aliased to generic macro MODIFY\_REG(), the first parameter is the ADC handle State, the second parameter is the bit field to clear, the third and last parameter is the bit field to set.

ADC\_CLEAR\_ERRORCODE

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: ADC handle

**Return value:**

- None

IS\_ADC\_CLOCKPRESCALER  
IS\_ADC\_SAMPLING\_DELAY  
IS\_ADC\_RESOLUTION  
IS\_ADC\_EXT\_TRIG\_EDGE  
IS\_ADC\_EXT\_TRIG  
IS\_ADC\_DATA\_ALIGN  
IS\_ADC\_SAMPLE\_TIME  
IS\_ADC\_EOCSelection  
IS\_ADC\_EVENT\_TYPE  
IS\_ADC\_ANALOG\_WATCHDOG  
IS\_ADC\_CHANNELS\_TYPE  
IS\_ADC\_THRESHOLD  
IS\_ADC\_REGULAR\_LENGTH  
IS\_ADC\_REGULAR\_RANK  
IS\_ADC\_REGULAR\_DISC\_NUMBER  
IS\_ADC\_RANGE  
ADC\_SQR1

**Description:**

- Set ADC Regular channel sequence length.

**Parameters:**

- \_NbrOfConversion\_: Regular channel sequence length.

**Return value:**

- None

ADC\_SMPR1

**Description:**

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

**Parameters:**

- \_SAMPLETIME\_: Sample time parameter.
- \_CHANNELNB\_: Channel number.

**Return value:**

- None

ADC\_SMPR2

**Description:**

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

**Parameters:**

- \_SAMPLETIME\_: Sample time parameter.

- `_CHANNELNB_`: Channel number.

**Description:**

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

**Parameters:**

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

**Return value:**

- None

`ADC_SQR3_RK`

**Description:**

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

**Parameters:**

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

**Return value:**

- None

`ADC_SQR1_RK`

**Description:**

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

**Parameters:**

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

**Return value:**

- None

`ADC_CR2_CONTINUOUS`

**Description:**

- Enable ADC continuous conversion mode.

**Parameters:**

- `_CONTINUOUS_MODE_`: Continuous mode.

**Return value:**

- None

`ADC_CR1_DISCONTINUOUS`

**Description:**

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

ADC\_CR1\_SCANCONV

**Parameters:**

- `_NBR_DISCONTINUOUSCONV_`: Number of discontinuous conversions.

**Return value:**

- None

**Description:**

- Enable ADC scan mode.

**Parameters:**

- `_SCANCONV_MODE_`: Scan conversion mode.

**Return value:**

- None

ADC\_CR2\_EOCSelection

**Description:**

- Enable the ADC end of conversion selection.

**Parameters:**

- `_EOCSelection_MODE_`: End of conversion selection mode.

**Return value:**

- None

ADC\_CR2\_DMAContReq

**Description:**

- Enable the ADC DMA continuous request.

**Parameters:**

- `_DMAContReq_MODE_`: DMA continuous request mode.

**Return value:**

- None

ADC\_GET\_RESOLUTION

**Description:**

- Return resolution bits in CR1 register.

**Parameters:**

- `__HANDLE__`: ADC handle

**Return value:**

- None

***ADC Resolution***

ADC\_RESOLUTION\_12B

ADC\_RESOLUTION\_10B

ADC\_RESOLUTION\_8B

ADC\_RESOLUTION\_6B

***ADC Sampling Times***

ADC\_SAMPLETIME\_3CYCLES

ADC\_SAMPLETIME\_15CYCLES

ADC\_SAMPLETIME\_28CYCLES

ADC\_SAMPLETIME\_56CYCLES

ADC\_SAMPLETIME\_84CYCLES

ADC\_SAMPLETIME\_112CYCLES

ADC\_SAMPLETIME\_144CYCLES

ADC\_SAMPLETIME\_480CYCLES

## 5 HAL ADC Extension Driver

### 5.1 ADCEx Firmware driver registers structures

#### 5.1.1 ADC\_InjectionConfTypeDef

##### Data Fields

- *uint32\_t InjectedChannel*
- *uint32\_t InjectedRank*
- *uint32\_t InjectedSamplingTime*
- *uint32\_t InjectedOffset*
- *uint32\_t InjectedNbrOfConversion*
- *uint32\_t InjectedDiscontinuousConvMode*
- *uint32\_t AutoInjectedConv*
- *uint32\_t ExternalTrigInjecConv*
- *uint32\_t ExternalTrigInjecConvEdge*

##### Field Documentation

- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedChannel***  
Selection of ADC channel to configure This parameter can be a value of **ADC\_channels** Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability.
- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedRank***  
Rank in the injected group sequencer This parameter must be a value of **ADCEx\_injected\_rank** Note: In case of need to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions can be adjusted)
- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedSamplingTime***  
Sampling time value to be set for the selected channel. Unit: ADC clock cycles  
Conversion time is the addition of sampling time and processing time (12 ADC clock cycles at ADC resolution 12 bits, 11 cycles at 10 bits, 9 cycles at 8 bits, 7 cycles at 6 bits). This parameter can be a value of **ADC\_sampling\_times** Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS\_vrefint, TS\_temp (values rough order: 4us min).
- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedOffset***  
Defines the offset to be subtracted from the raw converted data (for channels set on injected group only). Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFFF, 0x3FF, 0xFF or 0x3F respectively.
- ***uint32\_t ADC\_InjectionConfTypeDef::InjectedNbrOfConversion***  
Specifies the number of ranks that will be converted within the injected group sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number

between Min\_Data = 1 and Max\_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL\_ADCEx\_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***uint32\_t ADC\_InjectionTypeDef::InjectedDiscontinuousConvMode***  
Specifies whether the conversions sequence of injected group is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE. Note: For injected group, number of discontinuous ranks increment is fixed to one-by-one. Caution: this setting impacts the entire injected group. Therefore, call of **HAL\_ADCEx\_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32\_t ADC\_InjectionTypeDef::AutoInjectedConv***  
Enables or disables the selected ADC automatic injected group conversion after regular one. This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE) Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC\_SOFTWARE\_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL\_ADCEx\_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32\_t ADC\_InjectionTypeDef::ExternalTrigInjecConv***  
Selects the external event used to trigger the conversion start of injected group. If set to ADC\_INJECTED\_SOFTWARE\_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of **ADCEx\_External\_trigger\_Source\_Injected**. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behaviour in case of another parameter update on the fly) Caution: this setting impacts the entire injected group. Therefore, call of **HAL\_ADCEx\_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32\_t ADC\_InjectionTypeDef::ExternalTrigInjecConvEdge***  
Selects the external trigger edge of injected group. This parameter can be a value of **ADCEx\_External\_trigger\_edge\_Injected**. If trigger is set to ADC\_INJECTED\_SOFTWARE\_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL\_ADCEx\_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

## 5.1.2 ADC\_MultiModeTypeDef

### 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 [\*ADC\\_delay\\_between\\_2\\_sampling\\_phases\*](#)

## 5.2 ADCEx Firmware driver API description

### 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 `__HAL_RCC_ADC_CLK_ENABLE()`
  - b. ADC pins configuration
    - Enable the clock for the ADC GPIOs using the following function:  
`__HAL_RCC_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  
`__HAL_RCC_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.

This section contains the following APIs:

- [\*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 HAL\_ADCEx\_InjectedStart

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStart(ADC_HandleTypeDef * hadc)</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>• HAL status</li> </ul>

### 5.2.4 HAL\_ADCEx\_InjectedStart\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT(ADC_HandleTypeDef * hadc)</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>• HAL status.</li> </ul>

### 5.2.5 HAL\_ADCEx\_InjectedStop

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStop(ADC_HandleTypeDef * hadc)</b>
Function Description	Stop conversion of injected channels.
Parameters	<ul style="list-style-type: none"> <li>• <b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.</li> <li>• If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.</li> <li>• In case of auto-injection mode, HAL_ADC_Stop must be used.</li> </ul>

### 5.2.6 HAL\_ADCEx\_InjectedPollForConversion

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)</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>HAL status</li> </ul>

### 5.2.7 HAL\_ADCEx\_InjectedStop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)</b>
Function Description	Stop conversion of injected channels, disable interruption of end-of-conversion.
Parameters	<ul style="list-style-type: none"> <li><b>hadc:</b> ADC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.</li> <li>If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.</li> </ul>

### 5.2.8 HAL\_ADCEx\_InjectedGetValue

Function Name	<b>uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef * hadc, uint32_t InjectedRank)</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: ADC_INJECTED_RANK_1: Injected Channel1 selectedADC_INJECTED_RANK_2: Injected Channel2 selectedADC_INJECTED_RANK_3: Injected Channel3 selectedADC_INJECTED_RANK_4: Injected Channel4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 5.2.9 HAL\_ADCEx\_MultiModeStart\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * 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> </ul>

- **Length:** The length of data to be transferred from ADC peripheral to memory.

Return values

- HAL status

Notes

- Caution: This function must be used only with the ADC master.

### 5.2.10 HAL\_ADCEx\_MultiModeStop\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA(ADC_HandleTypeDef * hadc)</code>
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>• HAL status</li> </ul>

### 5.2.11 HAL\_ADCEx\_MultiModeGetValue

Function Name	<code>uint32_t HAL_ADCEx_MultiModeGetValue(ADC_HandleTypeDef * hadc)</code>
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>• The converted data value.</li> </ul>

### 5.2.12 HAL\_ADCEx\_InjectedConvCpltCallback

Function Name	<code>void HAL_ADCEx_InjectedConvCpltCallback(ADC_HandleTypeDef * hadc)</code>
Function Description	Injected conversion complete 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>

### 5.2.13 HAL\_ADCEx\_InjectedConfigChannel

Function Name	<code>HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel(ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)</code>
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 ADC_HandleTypeDef 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>

### 5.2.14 HAL\_ADCEx\_MultiModeConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel(ADC_HandleTypeDef * hadc, ADC_MultiModeTypeDef * 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>• HAL status</li> </ul>

## 5.3 ADCEx Firmware driver defines

### 5.3.1 ADCEx

#### *ADC Specific Channels*

ADC\_CHANNEL\_TEMPSENSOR

#### *ADC Common Mode*

ADC\_MODE\_INDEPENDENT

ADC\_DUALMODE\_REGSIMULT\_INJECSIMULT

ADC\_DUALMODE\_REGSIMULT.AlterTrig

ADC\_DUALMODE\_INJECSIMULT

ADC\_DUALMODE\_REGSIMULT

ADC\_DUALMODE\_INTERL

ADC\_DUALMODE.AlterTrig

ADC\_TRIPLEMODE\_REGSIMULT\_INJECSIMULT

ADC\_TRIPLEMODE\_REGSIMULT.AlterTrig

ADC\_TRIPLEMODE\_INJECSIMULT

ADC\_TRIPLEMODE\_REGSIMULT

ADC\_TRIPLEMODE\_INTERL

ADC\_TRIPLEMODE.AlterTrig

#### *ADC Direct Memory Access Mode For Multi Mode*

ADC\_DMAACCESSMODE\_DISABLED DMA mode disabled

ADC\_DMAACCESSMODE\_1 DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)

ADC\_DMAACCESSMODE\_2 DMA mode 2 enabled (2 / 3 half-words by pairs - 2&1 then 1&3 then 3&2)

ADC\_DMAACCESSMODE\_3 DMA mode 3 enabled (2 / 3 bytes by pairs - 2&1 then 1&3 then 3&2)

#### *ADC External Trigger Edge Injected*

ADC\_EXTERNALTRIGINJECCONVEDGE\_NONE  
ADC\_EXTERNALTRIGINJECCONVEDGE\_RISING  
ADC\_EXTERNALTRIGINJECCONVEDGE\_FALLING  
ADC\_EXTERNALTRIGINJECCONVEDGE\_RISINGFALLING

***ADC External Trigger Source Injected***

ADC\_EXTERNALTRIGINJECCONV\_T1\_CC4  
ADC\_EXTERNALTRIGINJECCONV\_T1\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T2\_CC1  
ADC\_EXTERNALTRIGINJECCONV\_T2\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T3\_CC2  
ADC\_EXTERNALTRIGINJECCONV\_T3\_CC4  
ADC\_EXTERNALTRIGINJECCONV\_T4\_CC1  
ADC\_EXTERNALTRIGINJECCONV\_T4\_CC2  
ADC\_EXTERNALTRIGINJECCONV\_T4\_CC3  
ADC\_EXTERNALTRIGINJECCONV\_T4\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T5\_CC4  
ADC\_EXTERNALTRIGINJECCONV\_T5\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T8\_CC2  
ADC\_EXTERNALTRIGINJECCONV\_T8\_CC3  
ADC\_EXTERNALTRIGINJECCONV\_T8\_CC4  
ADC\_EXTERNALTRIGINJECCONV\_EXT\_IT15  
ADC\_INJECTED\_SOFTWARE\_START

***ADC Injected Rank***

ADC\_INJECTED\_RANK\_1  
ADC\_INJECTED\_RANK\_2  
ADC\_INJECTED\_RANK\_3  
ADC\_INJECTED\_RANK\_4

***ADC Private Macros***

IS\_ADC\_CHANNEL  
IS\_ADC\_MODE  
IS\_ADC\_DMA\_ACCESS\_MODE  
IS\_ADC\_EXT\_INJEC\_TRIG\_EDGE  
IS\_ADC\_EXT\_INJEC\_TRIG  
IS\_ADC\_INJECTED\_LENGTH  
IS\_ADC\_INJECTED\_RANK  
ADC\_JSQR

**Description:**

- Set the selected injected Channel rank.

**Parameters:**

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

**Return value:**

- None

## 6 HAL CAN Generic Driver

### 6.1 CAN Firmware driver registers structures

#### 6.1.1 CAN\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Mode*
- *uint32\_t SJW*
- *uint32\_t BS1*
- *uint32\_t BS2*
- *uint32\_t TTCM*
- *uint32\_t ABOM*
- *uint32\_t AWUM*
- *uint32\_t NART*
- *uint32\_t RFLM*
- *uint32\_t TXFP*

##### Field Documentation

- ***uint32\_t CAN\_InitTypeDef::Prescaler***  
Specifies the length of a time quantum. This parameter must be a number between Min\_Data = 1 and Max\_Data = 1024
- ***uint32\_t CAN\_InitTypeDef::Mode***  
Specifies the CAN operating mode. This parameter can be a value of [CAN\\_operating\\_mode](#)
- ***uint32\_t CAN\_InitTypeDef::SJW***  
Specifies the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter can be a value of [CAN\\_synchronisation\\_jump\\_width](#)
- ***uint32\_t CAN\_InitTypeDef::BS1***  
Specifies the number of time quanta in Bit Segment 1. This parameter can be a value of [CAN\\_time\\_quantum\\_in\\_bit\\_segment\\_1](#)
- ***uint32\_t CAN\_InitTypeDef::BS2***  
Specifies the number of time quanta in Bit Segment 2. This parameter can be a value of [CAN\\_time\\_quantum\\_in\\_bit\\_segment\\_2](#)
- ***uint32\_t CAN\_InitTypeDef::TTCM***  
Enable or disable the time triggered communication mode. This parameter can be set to ENABLE or DISABLE.
- ***uint32\_t CAN\_InitTypeDef::ABOM***  
Enable or disable the automatic bus-off management. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t CAN\_InitTypeDef::AWUM***  
Enable or disable the automatic wake-up mode. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t CAN\_InitTypeDef::NART***  
Enable or disable the non-automatic retransmission mode. This parameter can be set to ENABLE or DISABLE

- ***uint32\_t CAN\_InitTypeDef::RFLM***  
Enable or disable the receive FIFO Locked mode. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t CAN\_InitTypeDef::TXFP***  
Enable or disable the transmit FIFO priority. This parameter can be set to ENABLE or DISABLE

## 6.1.2 CAN\_FilterConfTypeDef

### Data Fields

- ***uint32\_t FilterIdHigh***
- ***uint32\_t FilterIdLow***
- ***uint32\_t FilterMaskIdHigh***
- ***uint32\_t FilterMaskIdLow***
- ***uint32\_t FilterFIFOAssignment***
- ***uint32\_t FilterNumber***
- ***uint32\_t FilterMode***
- ***uint32\_t FilterScale***
- ***uint32\_t FilterActivation***
- ***uint32\_t BankNumber***

### Field Documentation

- ***uint32\_t CAN\_FilterConfTypeDef::FilterIdHigh***  
Specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t CAN\_FilterConfTypeDef::FilterIdLow***  
Specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t CAN\_FilterConfTypeDef::FilterMaskIdHigh***  
Specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t CAN\_FilterConfTypeDef::FilterMaskIdLow***  
Specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t CAN\_FilterConfTypeDef::FilterFIFOAssignment***  
Specifies the FIFO (0 or 1) which will be assigned to the filter. This parameter can be a value of [CAN\\_filter\\_FIFO](#)
- ***uint32\_t CAN\_FilterConfTypeDef::FilterNumber***  
Specifies the filter which will be initialized. This parameter must be a number between Min\_Data = 0 and Max\_Data = 27
- ***uint32\_t CAN\_FilterConfTypeDef::FilterMode***  
Specifies the filter mode to be initialized. This parameter can be a value of [CAN\\_filter\\_mode](#)
- ***uint32\_t CAN\_FilterConfTypeDef::FilterScale***  
Specifies the filter scale. This parameter can be a value of [CAN\\_filter\\_scale](#)

- ***uint32\_t CAN\_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

### 6.1.3 CanTxMsgTypeDef

#### Data Fields

- ***uint32\_t StdId***
- ***uint32\_t ExtId***
- ***uint32\_t IDE***
- ***uint32\_t RTR***
- ***uint32\_t DLC***
- ***uint8\_t Data***

#### Field Documentation

- ***uint32\_t CanTxMsgTypeDef::StdId***  
Specifies the standard identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x7FF
- ***uint32\_t CanTxMsgTypeDef::ExtId***  
Specifies the extended identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x1FFFFFFF
- ***uint32\_t CanTxMsgTypeDef::IDE***  
Specifies the type of identifier for the message that will be transmitted. This parameter can be a value of [CAN\\_Identifier\\_Type](#)
- ***uint32\_t CanTxMsgTypeDef::RTR***  
Specifies the type of frame for the message that will be transmitted. This parameter can be a value of [CAN\\_remote\\_transmission\\_request](#)
- ***uint32\_t CanTxMsgTypeDef::DLC***  
Specifies the length of the frame that will be transmitted. This parameter must be a number between Min\_Data = 0 and Max\_Data = 8
- ***uint8\_t CanTxMsgTypeDef::Data[8]***  
Contains the data to be transmitted. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0xFF

### 6.1.4 CanRxMsgTypeDef

#### Data Fields

- ***uint32\_t StdId***
- ***uint32\_t ExtId***
- ***uint32\_t IDE***
- ***uint32\_t RTR***
- ***uint32\_t DLC***
- ***uint8\_t Data***
- ***uint32\_t FMI***

- *uint32\_t FIFONumber*

#### Field Documentation

- ***uint32\_t CanRxMsgTypeDef::StdId***  
Specifies the standard identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x7FF
- ***uint32\_t CanRxMsgTypeDef::ExtId***  
Specifies the extended identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x1FFFFFFF
- ***uint32\_t CanRxMsgTypeDef::IDE***  
Specifies the type of identifier for the message that will be received. This parameter can be a value of [CAN\\_Identifier\\_Type](#)
- ***uint32\_t CanRxMsgTypeDef::RTR***  
Specifies the type of frame for the received message. This parameter can be a value of [CAN\\_remote\\_transmission\\_request](#)
- ***uint32\_t CanRxMsgTypeDef::DLC***  
Specifies the length of the frame that will be received. This parameter must be a number between Min\_Data = 0 and Max\_Data = 8
- ***uint8\_t CanRxMsgTypeDef::Data[8]***  
Contains the data to be received. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0xFF
- ***uint32\_t CanRxMsgTypeDef::FMI***  
Specifies the index of the filter the message stored in the mailbox passes through. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0xFF
- ***uint32\_t CanRxMsgTypeDef::FIFONumber***  
Specifies the receive FIFO number. This parameter can be CAN\_FIFO0 or CAN\_FIFO1

## 6.1.5 CAN\_HandleTypeDef

#### 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

## 6.2 CAN Firmware driver API description

### 6.2.1 How to use this driver

1. Enable the CAN controller interface clock using `_HAL_RCC_CAN1_CLK_ENABLE()` for CAN1 and `_HAL_RCC_CAN2_CLK_ENABLE()` for CAN2 In case you are using CAN2 only, you have to enable the CAN1 clock.
2. CAN pins configuration
  - Enable the clock for the CAN GPIOs using the following function:  
`_GPIOx_CLK_ENABLE()`
  - Connect and configure the involved CAN pins to AF9 using the following function  
`HAL_GPIO_Init()`
3. Initialize 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.

This section contains the following APIs:

- [`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.

This section contains the following APIs:

- [`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

This section contains the following APIs:

- [`HAL\_CAN\_GetState\(\)`](#)
- [`HAL\_CAN\_GetError\(\)`](#)

**6.2.5 HAL\_CAN\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_CAN_Init (CAN_HandleTypeDef * hcan)</b>
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>• HAL status</li> </ul>

**6.2.6 HAL\_CAN\_ConfigFilter**

Function Name	<b>HAL_StatusTypeDef HAL_CAN_ConfigFilter (CAN_HandleTypeDef * hcan, CAN_FilterConfTypeDef * 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>

**6.2.7 HAL\_CAN\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_CAN_DeInit (CAN_HandleTypeDef * 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>• HAL status</li> </ul>

**6.2.8 HAL\_CAN\_MspInit**

Function Name	<b>void HAL_CAN_MspInit (CAN_HandleTypeDef * 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>

**6.2.9 HAL\_CAN\_MspDeInit**

Function Name	<b>void HAL_CAN_MspDeInit (CAN_HandleTypeDef * 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>
<b>6.2.10</b>	<b>HAL_CAN_Transmit</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CAN_Transmit (CAN_HandleTypeDef * 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>HAL status</li> </ul>	
<b>6.2.11</b>	<b>HAL_CAN_Transmit_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CAN_Transmit_IT (CAN_HandleTypeDef * 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>HAL status</li> </ul>	
<b>6.2.12</b>	<b>HAL_CAN_Receive</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CAN_Receive (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>HAL status</li> </ul>	
<b>6.2.13</b>	<b>HAL_CAN_Receive_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CAN_Receive_IT (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>HAL status</li> </ul>	
<b>6.2.14</b>	<b>HAL_CAN_Sleep</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CAN_Sleep (CAN_HandleTypeDef * 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>• HAL status.</li> </ul>

### 6.2.15 HAL\_CAN\_WakeUp

Function Name	<b>HAL_StatusTypeDef HAL_CAN_WakeUp (CAN_HandleTypeDef * 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>• HAL status.</li> </ul>

### 6.2.16 HAL\_CAN\_IRQHandler

Function Name	<b>void HAL_CAN_IRQHandler (CAN_HandleTypeDef * 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>

### 6.2.17 HAL\_CAN\_TxCpltCallback

Function Name	<b>void HAL_CAN_TxCpltCallback (CAN_HandleTypeDef * 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>

### 6.2.18 HAL\_CAN\_RxCpltCallback

Function Name	<b>void HAL_CAN_RxCpltCallback (CAN_HandleTypeDef * 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>

### 6.2.19 HAL\_CAN\_ErrorCallback

Function Name	<b>void HAL_CAN_ErrorCallback (CAN_HandleTypeDef * 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>

## 6.2.20 HAL\_CAN\_GetState

Function Name	<b>HAL_CAN_StateTypeDef HAL_CAN_GetState (CAN_HandleTypeDef * hcan)</b>
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>• HAL state</li> </ul>

## 6.2.21 HAL\_CAN\_GetError

Function Name	<b>uint32_t HAL_CAN_GetError (CAN_HandleTypeDef * hcan)</b>
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>• CAN Error Code</li> </ul>

## 6.3 CAN Firmware driver defines

### 6.3.1 CAN

#### *CAN Exported Macros*

<b>_HAL_CAN_RESET_HANDLE_STATE</b>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Reset CAN handle state.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <b>_HANDLE_</b>: specifies the CAN Handle.</li> </ul>
<b>_HAL_CAN_ENABLE_IT</b>	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>
	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Enable the specified CAN interrupts.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <b>_HANDLE_</b>: CAN handle</li> <li>• <b>_INTERRUPT_</b>: CAN Interrupt</li> </ul>
<b>_HAL_CAN_DISABLE_IT</b>	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>
	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Disable the specified CAN interrupts.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <b>_HANDLE_</b>: CAN handle</li> <li>• <b>_INTERRUPT_</b>: CAN Interrupt</li> </ul>
	<b>Return value:</b>

- None

### \_HAL\_CAN\_MSG\_PENDING

**Description:**

- Return the number of pending received messages.

**Parameters:**

- \_HANDLE\_: CAN handle
- \_FIFONUMBER\_: Receive FIFO number, CAN\_FIFO0 or CAN\_FIFO1.

**Return value:**

- The: number of pending message.

### \_HAL\_CAN\_GET\_FLAG

**Description:**

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

**Parameters:**

- \_HANDLE\_: CAN Handle
- \_FLAG\_: specifies the flag to check. This parameter can be one of the following values:
  - CAN\_TSR\_RQCP0: Request MailBox0 Flag
  - CAN\_TSR\_RQCP1: Request MailBox1 Flag
  - CAN\_TSR\_RQCP2: Request MailBox2 Flag
  - CAN\_FLAG\_TXOK0: Transmission OK MailBox0 Flag
  - CAN\_FLAG\_TXOK1: Transmission OK MailBox1 Flag
  - CAN\_FLAG\_TXOK2: Transmission OK MailBox2 Flag
  - CAN\_FLAG\_TME0: Transmit mailbox 0 empty Flag
  - CAN\_FLAG\_TME1: Transmit mailbox 1 empty Flag
  - CAN\_FLAG\_TME2: Transmit mailbox 2 empty Flag
  - CAN\_FLAG\_FMP0: FIFO 0 Message Pending Flag
  - CAN\_FLAG\_FF0: FIFO 0 Full Flag
  - CAN\_FLAG\_FOV0: FIFO 0 Overrun Flag
  - CAN\_FLAG\_FMP1: FIFO 1 Message Pending Flag
  - CAN\_FLAG\_FF1: FIFO 1 Full Flag
  - CAN\_FLAG\_FOV1: FIFO 1 Overrun Flag
  - CAN\_FLAG\_WKU: Wake up Flag
  - CAN\_FLAG\_SLAK: Sleep acknowledge Flag

- CAN\_FLAG\_SLAKI: Sleep acknowledge Flag
- CAN\_FLAG\_EWG: Error Warning Flag
- CAN\_FLAG\_EPV: Error Passive Flag
- CAN\_FLAG\_BOF: Bus-Off Flag

**Return value:**

- The new state of \_\_FLAG\_\_ (TRUE or FALSE).

**\_HAL\_CAN\_CLEAR\_FLAG****Description:**

- Clear the specified CAN pending flag.

**Parameters:**

- \_\_HANDLE\_\_: CAN Handle.
- \_\_FLAG\_\_: specifies the flag to check.  
This parameter can be one of the following values:
  - CAN\_TSR\_RQCP0: Request MailBox0 Flag
  - CAN\_TSR\_RQCP1: Request MailBox1 Flag
  - CAN\_TSR\_RQCP2: Request MailBox2 Flag
  - CAN\_FLAG\_TXOK0: Transmission OK MailBox0 Flag
  - CAN\_FLAG\_TXOK1: Transmission OK MailBox1 Flag
  - CAN\_FLAG\_TXOK2: Transmission OK MailBox2 Flag
  - CAN\_FLAG\_TME0: Transmit mailbox 0 empty Flag
  - CAN\_FLAG\_TME1: Transmit mailbox 1 empty Flag
  - CAN\_FLAG\_TME2: Transmit mailbox 2 empty Flag
  - CAN\_FLAG\_FMP0: FIFO 0 Message Pending Flag
  - CAN\_FLAG\_FF0: FIFO 0 Full Flag
  - CAN\_FLAG\_FOV0: FIFO 0 Overrun Flag
  - CAN\_FLAG\_FMP1: FIFO 1 Message Pending Flag
  - CAN\_FLAG\_FF1: FIFO 1 Full Flag
  - CAN\_FLAG\_FOV1: FIFO 1 Overrun Flag
  - CAN\_FLAG\_WKU: Wake up Flag
  - CAN\_FLAG\_SLAK: Sleep acknowledge Flag
  - CAN\_FLAG\_SLAKI: Sleep acknowledge Flag
  - CAN\_FLAG\_EWG: Error Warning Flag

- CAN\_FLAG\_EPV: Error Passive Flag
- CAN\_FLAG\_BOF: Bus-Off Flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

**\_\_HAL\_CAN\_GET\_IT\_SOURCE****Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: CAN Handle
- \_\_INTERRUPT\_\_: specifies the CAN interrupt source to check. This parameter can be one of the following values:
  - CAN\_IT\_TME: Transmit mailbox empty interrupt enable
  - CAN\_IT\_FMP0: FIFO0 message pending interrupt enable
  - CAN\_IT\_FMP1: FIFO1 message pending interrupt enable

**Return value:**

- The: new state of \_\_IT\_\_ (TRUE or FALSE).

**\_\_HAL\_CAN\_TRANSMIT\_STATUS****Description:**

- Check the transmission status of a CAN Frame.

**Parameters:**

- \_\_HANDLE\_\_: CAN Handle
- \_\_TRANSMITMAILBOX\_\_: the number of the mailbox that is used for transmission.

**Return value:**

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

**\_\_HAL\_CAN\_FIFO\_RELEASE****Description:**

- Release the specified receive FIFO.

**Parameters:**

- \_\_HANDLE\_\_: CAN handle
- \_\_FIFONUMBER\_\_: Receive FIFO number, CAN\_FIFO0 or CAN\_FIFO1.

**Return value:**

- None

**\_\_HAL\_CAN\_CANCEL\_TRANSMIT****Description:**

- Cancel a transmit request.

**Parameters:**

- HANDLE: CAN Handle
- TRANSMITMAILBOX: the number of the mailbox that is used for transmission.

**Return value:**

- None

\_HAL\_CAN\_DBG\_FREEZE**Description:**

- Enable or disable the DBG Freeze for CAN.

**Parameters:**

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

**Return value:**

- None

**CAN Filter FIFO**

CAN\_FILTER\_FIFO0 Filter FIFO 0 assignment for filter x

CAN\_FILTER\_FIFO1 Filter FIFO 1 assignment for filter x

**CAN Filter Mode**

CAN\_FILTERMODE\_IDMASK Identifier mask mode

CAN\_FILTERMODE\_IDLIST Identifier list mode

**CAN Filter Scale**

CAN\_FILTERSCALE\_16BIT Two 16-bit filters

CAN\_FILTERSCALE\_32BIT One 32-bit filter

**CAN Flags**

CAN\_FLAG\_RQCP0 Request MailBox0 flag

CAN\_FLAG\_RQCP1 Request MailBox1 flag

CAN\_FLAG\_RQCP2 Request MailBox2 flag

CAN\_FLAG\_TXOK0 Transmission OK MailBox0 flag

CAN\_FLAG\_TXOK1 Transmission OK MailBox1 flag

CAN\_FLAG\_TXOK2 Transmission OK MailBox2 flag

CAN\_FLAG\_TME0 Transmit mailbox 0 empty flag

CAN\_FLAG\_TME1 Transmit mailbox 0 empty flag

CAN\_FLAG\_TME2 Transmit mailbox 0 empty flag

CAN\_FLAG\_FF0 FIFO 0 Full flag

---

CAN_FLAG_FOV0	FIFO 0 Overrun flag
CAN_FLAG_FF1	FIFO 1 Full flag
CAN_FLAG_FOV1	FIFO 1 Overrun flag
CAN_FLAG_WKU	Wake up flag
CAN_FLAG_SLAK	Sleep acknowledge flag
CAN_FLAG_SLAKI	Sleep acknowledge flag
CAN_FLAG_EWG	Error warning flag
CAN_FLAG_EPV	Error passive flag
CAN_FLAG_BOF	Bus-Off flag

***CAN Identifier Type***

CAN_ID_STD	Standard Id
CAN_ID_EXT	Extended Id

***CAN InitStatus***

CAN_INITSTATUS_FAILED	CAN initialization failed
CAN_INITSTATUS_SUCCESS	CAN initialization OK

***CAN Interrupts***

CAN_IT_TME	Transmit mailbox empty interrupt
CAN_IT_FMP0	FIFO 0 message pending interrupt
CAN_IT_FF0	FIFO 0 full interrupt
CAN_IT_FOV0	FIFO 0 overrun interrupt
CAN_IT_FMP1	FIFO 1 message pending interrupt
CAN_IT_FF1	FIFO 1 full interrupt
CAN_IT_FOV1	FIFO 1 overrun interrupt
CAN_IT_WKU	Wake-up interrupt
CAN_IT_SLK	Sleep acknowledge interrupt
CAN_IT_EWG	Error warning interrupt
CAN_IT_EPV	Error passive interrupt
CAN_IT_BOF	Bus-off interrupt
CAN_IT_LEC	Last error code interrupt
CAN_IT_ERR	Error Interrupt

***CAN Mailboxes Definition***

CAN_TXMAILBOX_0
CAN_TXMAILBOX_1
CAN_TXMAILBOX_2

***CAN Operating Mode***

CAN_MODE_NORMAL	Normal mode
-----------------	-------------

CAN_MODE_LOOPBACK	Loopback mode
CAN_MODE_SILENT	Silent mode
CAN_MODE_SILENT_LOOPBACK	Loopback combined with silent mode
<b>CAN Private Constants</b>	
CAN_TIMEOUT_VALUE	
CAN_TXSTATUS_NOMAILBOX	CAN cell did not provide CAN_TxStatus_NoMailBox
CAN_FLAG_MASK	
<b>CAN Private Macros</b>	
IS_CAN_MODE	
IS_CAN_SJW	
IS_CAN_BS1	
IS_CAN_BS2	
IS_CAN_PRESCALER	
IS_CAN_FILTER_NUMBER	
IS_CAN_FILTER_MODE	
IS_CAN_FILTER_SCALE	
IS_CAN_FILTER_FIFO	
IS_CAN_BANKNUMBER	
IS_CAN_TRANSMITMAILBOX	
IS_CAN_STID	
IS_CAN_EXTID	
IS_CAN_DLC	
IS_CAN_IDTYPE	
IS_CAN_RTR	
IS_CAN_FIFO	
<b>CAN Receive FIFO Number Constants</b>	
CAN_FIFO0	CAN FIFO 0 used to receive
CAN_FIFO1	CAN FIFO 1 used to receive
<b>CAN Remote Transmission Request</b>	
CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame
<b>CAN Synchronisation Jump Width</b>	
CAN_SJW_1TQ	1 time quantum
CAN_SJW_2TQ	2 time quantum
CAN_SJW_3TQ	3 time quantum
CAN_SJW_4TQ	4 time quantum
<b>CAN Time Quantum in bit segment 1</b>	

---

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

***CAN Time Quantum in bit segment 2***

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

## 7 HAL CORTEX Generic Driver

### 7.1 CORTEX Firmware driver registers structures

#### 7.1.1 MPU\_Region\_InitTypeDef

##### Data Fields

- *uint8\_t Enable*
- *uint8\_t Number*
- *uint32\_t BaseAddress*
- *uint8\_t Size*
- *uint8\_t SubRegionDisable*
- *uint8\_t TypeExtField*
- *uint8\_t AccessPermission*
- *uint8\_t DisableExec*
- *uint8\_t IsShareable*
- *uint8\_t IsCacheable*
- *uint8\_t IsBufferable*

##### Field Documentation

- ***uint8\_t MPU\_Region\_InitTypeDef::Enable***  
Specifies the status of the region. This parameter can be a value of **CORTEX MPU Region Enable**
- ***uint8\_t MPU\_Region\_InitTypeDef::Number***  
Specifies the number of the region to protect. This parameter can be a value of **CORTEX MPU Region Number**
- ***uint32\_t MPU\_Region\_InitTypeDef::BaseAddress***  
Specifies the base address of the region to protect.
- ***uint8\_t MPU\_Region\_InitTypeDef::Size***  
Specifies the size of the region to protect. This parameter can be a value of **CORTEX MPU Region Size**
- ***uint8\_t MPU\_Region\_InitTypeDef::SubRegionDisable***  
Specifies the number of the subregion protection to disable. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF
- ***uint8\_t MPU\_Region\_InitTypeDef::TypeExtField***  
Specifies the TEX field level. This parameter can be a value of **CORTEX MPU TEX Levels**
- ***uint8\_t MPU\_Region\_InitTypeDef::AccessPermission***  
Specifies the region access permission type. This parameter can be a value of **CORTEX MPU Region Permission Attributes**
- ***uint8\_t MPU\_Region\_InitTypeDef::DisableExec***  
Specifies the instruction access status. This parameter can be a value of **CORTEX MPU Instruction Access**
- ***uint8\_t MPU\_Region\_InitTypeDef::IsShareable***  
Specifies the shareability status of the protected region. This parameter can be a value of **CORTEX MPU Access Shareable**

- ***uint8\_t MPU\_Region\_InitTypeDef::IsCacheable***  
Specifies the cacheable status of the region protected. This parameter can be a value of **CORTEX\_MPUMemoryAccessCacheable**
- ***uint8\_t MPU\_Region\_InitTypeDef::IsBufferable***  
Specifies the bufferable status of the protected region. This parameter can be a value of **CORTEX\_MPUMemoryAccessBufferable**

## 7.2 CORTEX Firmware driver API description

### 7.2.1 How to use this driver

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

This section provides functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M3 exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using HAL\_NVIC\_SetPriorityGrouping() function according to the following table.
2. Configure the priority of the selected IRQ Channels using HAL\_NVIC\_SetPriority().
3. Enable the selected IRQ Channels using HAL\_NVIC\_EnableIRQ().
4. please refer to programing manual for details in how to configure priority. When the NVIC\_PRIORITYGROUP\_0 is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the sub priority. IRQ priority order (sorted by highest to lowest priority): Lowest preemption priority Lowest sub priority Lowest hardware priority (IRQ number)

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

Setup SysTick Timer for time base.

- The HAL\_SYSTICK\_Config() function calls the SysTick\_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x0F).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK\_Div8 by calling the macro `__HAL_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 `stm32f2xx_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

## 7.2.2 Initialization and de-initialization functions

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

This section contains the following APIs:

- [\*HAL\\_NVIC\\_SetPriorityGrouping\(\)\*](#)
- [\*HAL\\_NVIC\\_SetPriority\(\)\*](#)
- [\*HAL\\_NVIC\\_EnableIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_DisableIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_SystemReset\(\)\*](#)
- [\*HAL\\_SYSTICK\\_Config\(\)\*](#)

## 7.2.3 Peripheral Control functions

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

This section contains the following APIs:

- [\*HAL\\_MPU\\_ConfigRegion\(\)\*](#)
- [\*HAL\\_NVIC\\_GetPriorityGrouping\(\)\*](#)
- [\*HAL\\_NVIC\\_GetPriority\(\)\*](#)
- [\*HAL\\_NVIC\\_SetPendingIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_GetPendingIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_ClearPendingIRQ\(\)\*](#)
- [\*HAL\\_NVIC\\_GetActive\(\)\*](#)
- [\*HAL\\_SYSTICK\\_CLKSourceConfig\(\)\*](#)
- [\*HAL\\_SYSTICK\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SYSTICK\\_Callback\(\)\*](#)

## 7.2.4 HAL\_NVIC\_SetPriorityGrouping

Function Name	<b>void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)</b>
Function Description	Sets the priority grouping field (preemption 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: NVIC_PRIORITYGROUP_0: 0 bits for preemption priority 4 bits for subpriority NVIC_PRIORITYGROUP_1: 1 bits for preemption priority 3 bits for subpriority NVIC_PRIORITYGROUP_2: 2 bits for preemption priority 2 bits for subpriority NVIC_PRIORITYGROUP_3: 3 bits for preemption priority 1 bits for subpriority NVIC_PRIORITYGROUP_4: 4 bits for preemption priority 0 bits for subpriority</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 preemption is no more possible. The pending IRQ priority will be managed only by the subpriority.</li> </ul>

## 7.2.5 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 the appropriate CMSIS device file (stm32f2xxxx.h))</li> <li><b>PreemptPriority:</b> The preemption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority</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>

### 7.2.6 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 the appropriate CMSIS device file (stm32f2xxxx.h))</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>

### 7.2.7 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 the appropriate CMSIS device file (stm32f2xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 7.2.8 HAL\_NVIC\_SystemReset

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

### 7.2.9 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>status - 0 Function succeeded. 1 Function failed.</li> </ul>

## 7.2.10 HAL\_MPU\_ConfigRegion

Function Name	<b>void HAL_MPU_ConfigRegion (MPU_Region_InitTypeDef * MPU_Init)</b>
Function Description	Initializes and configures the Region and the memory to be protected.
Parameters	<ul style="list-style-type: none"> <li><b>MPU_Init:</b> Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

## 7.2.11 HAL\_NVIC\_GetPriorityGrouping

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

## 7.2.12 HAL\_NVIC\_GetPriority

Function Name	<b>void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)</b>
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 the appropriate CMSIS device file (stm32f2xxxx.h))</li> <li><b>PriorityGroup:</b> the priority grouping bits length. This parameter can be one of the following values: NVIC_PRIORITYGROUP_0: 0 bits for preemption priority 4 bits for subpriority NVIC_PRIORITYGROUP_1: 1 bits for preemption priority 3 bits for subpriority NVIC_PRIORITYGROUP_2: 2 bits for preemption priority 2 bits for subpriority NVIC_PRIORITYGROUP_3: 3 bits for preemption priority 1 bits for subpriority NVIC_PRIORITYGROUP_4: 4 bits for preemption priority 0 bits for subpriority</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>

## 7.2.13 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 the appropriate CMSIS device file (stm32f2xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 7.2.14 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 the appropriate CMSIS device file (stm32f2xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• status - 0 Interrupt status is not pending. 1 Interrupt status is pending.</li> </ul>

#### 7.2.15 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 the appropriate CMSIS device file (stm32f2xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 7.2.16 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 the appropriate CMSIS device file (stm32f2xxxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• status - 0 Interrupt status is not pending. 1 Interrupt status is pending.</li> </ul>

#### 7.2.17 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</li> </ul>

parameter can be one of the following values:  
**SYSTICK\_CLKSOURCE\_HCLK\_DIV8**: AHB clock divided by 8 selected as SysTick clock source.  
**SYSTICK\_CLKSOURCE\_HCLK**: AHB clock selected as SysTick clock source.

Return values • None

### 7.2.18 HAL\_SYSTICK\_IRQHandler

Function Name **void HAL\_SYSTICK\_IRQHandler (void )**  
 Function Description This function handles SYSTICK interrupt request.  
 Return values • None

### 7.2.19 HAL\_SYSTICK\_Callback

Function Name **void HAL\_SYSTICK\_Callback (void )**  
 Function Description SYSTICK callback.  
 Return values • None

## 7.3 CORTEX Firmware driver defines

### 7.3.1 CORTEX

#### **CORTEX Exported Macros**

**\_\_HAL\_CORTEX\_SYSTICKCLK\_CON FIG** **Description:**  
 • Configures the SysTick clock source.  
**Parameters:**  
 • **\_\_CLKSRC**: specifies the SysTick clock source. This parameter can be one of the following values:  
   – **SYSTICK\_CLKSOURCE\_HCLK\_DIV8**: AHB clock divided by 8 selected as SysTick clock source.  
   – **SYSTICK\_CLKSOURCE\_HCLK**: AHB clock selected as SysTick clock source.  
**Return value:**  
 • None

#### **CORTEX MPU Instruction Access Bufferable**

**MPU\_ACCESS\_BUFFERABLE**

**MPU\_ACCESS\_NOT\_BUFFERABLE**

#### **CORTEX MPU Instruction Access Cacheable**

**MPU\_ACCESS\_CACHEABLE**

**MPU\_ACCESS\_NOT\_CACHEABLE**

#### **CORTEX MPU Instruction Access Shareable**

MPU\_ACCESS\_SHAREABLE  
MPU\_ACCESS\_NOT\_SHAREABLE  
**MPU HFNMI and PRIVILEGED Access control**  
MPU\_HFNMI\_PRIVDEF\_NONE  
MPU\_HARDFAULT\_NMI  
MPU\_PRIVILEGED\_DEFAULT  
MPU\_HFNMI\_PRIVDEF  
**CORTEX MPU Instruction Access**  
MPU\_INSTRUCTION\_ACCESS\_ENABLE  
MPU\_INSTRUCTION\_ACCESS\_DISABLE  
**CORTEX MPU Region Enable**  
MPU\_REGION\_ENABLE  
MPU\_REGION\_DISABLE  
**CORTEX MPU Region Number**  
MPU\_REGION\_NUMBER0  
MPU\_REGION\_NUMBER1  
MPU\_REGION\_NUMBER2  
MPU\_REGION\_NUMBER3  
MPU\_REGION\_NUMBER4  
MPU\_REGION\_NUMBER5  
MPU\_REGION\_NUMBER6  
MPU\_REGION\_NUMBER7  
**CORTEX MPU Region Permission Attributes**  
MPU\_REGION\_NO\_ACCESS  
MPU\_REGION\_PRIV\_RW  
MPU\_REGION\_PRIV\_RW\_URO  
MPU\_REGION\_FULL\_ACCESS  
MPU\_REGION\_PRIV\_RO  
MPU\_REGION\_PRIV\_RO\_URO  
**CORTEX MPU Region Size**  
MPU\_REGION\_SIZE\_32B  
MPU\_REGION\_SIZE\_64B  
MPU\_REGION\_SIZE\_128B  
MPU\_REGION\_SIZE\_256B  
MPU\_REGION\_SIZE\_512B  
MPU\_REGION\_SIZE\_1KB

MPU\_REGION\_SIZE\_2KB  
MPU\_REGION\_SIZE\_4KB  
MPU\_REGION\_SIZE\_8KB  
MPU\_REGION\_SIZE\_16KB  
MPU\_REGION\_SIZE\_32KB  
MPU\_REGION\_SIZE\_64KB  
MPU\_REGION\_SIZE\_128KB  
MPU\_REGION\_SIZE\_256KB  
MPU\_REGION\_SIZE\_512KB  
MPU\_REGION\_SIZE\_1MB  
MPU\_REGION\_SIZE\_2MB  
MPU\_REGION\_SIZE\_4MB  
MPU\_REGION\_SIZE\_8MB  
MPU\_REGION\_SIZE\_16MB  
MPU\_REGION\_SIZE\_32MB  
MPU\_REGION\_SIZE\_64MB  
MPU\_REGION\_SIZE\_128MB  
MPU\_REGION\_SIZE\_256MB  
MPU\_REGION\_SIZE\_512MB  
MPU\_REGION\_SIZE\_1GB  
MPU\_REGION\_SIZE\_2GB  
MPU\_REGION\_SIZE\_4GB

***MPU TEX Levels***

MPU\_TEX\_LEVEL0  
MPU\_TEX\_LEVEL1  
MPU\_TEX\_LEVEL2

***CORTEX Preemption Priority Group***

NVIC_PRIORITYGROUP_0	0 bits for pre-emption priority 4 bits for subpriority
NVIC_PRIORITYGROUP_1	1 bits for pre-emption priority 3 bits for subpriority
NVIC_PRIORITYGROUP_2	2 bits for pre-emption priority 2 bits for subpriority
NVIC_PRIORITYGROUP_3	3 bits for pre-emption priority 1 bits for subpriority
NVIC_PRIORITYGROUP_4	4 bits for pre-emption priority 0 bits for subpriority

***CORTEX Private Macros***

IS\_NVIC\_PRIORITY\_GROUP  
IS\_NVIC\_PREEMPTION\_PRIORITY  
IS\_NVIC\_SUB\_PRIORITY

IS\_NVIC\_DEVICE\_IRQ  
IS\_SYSTICK\_CLK\_SOURCE  
IS MPU\_REGION\_ENABLE  
IS MPU\_INSTRUCTION\_ACCESS  
IS MPU\_ACCESS\_SHAREABLE  
IS MPU\_ACCESS\_CACHEABLE  
IS MPU\_ACCESS\_BUFFERABLE  
IS MPU\_TEX\_LEVEL  
IS MPU\_REGION\_PERMISSION\_ATTRIBUTE  
IS MPU\_REGION\_NUMBER  
IS MPU\_REGION\_SIZE  
IS MPU\_SUB\_REGION\_DISABLE  
**CORTEX\_SysTick clock source**  
SYSTICK\_CLKSOURCE\_HCLK\_DIV8  
SYSTICK\_CLKSOURCE\_HCLK

## 8 HAL CRC Generic Driver

### 8.1 CRC Firmware driver registers structures

#### 8.1.1 CRC\_HandleTypeDef

##### 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

### 8.2 CRC Firmware driver API description

#### 8.2.1 How to use this driver

The CRC HAL driver can be used as follows:

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

#### 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

This section contains the following APIs:

- [`HAL\_CRC\_Init\(\)`](#)
- [`HAL\_CRC\_DeInit\(\)`](#)
- [`HAL\_CRC\_MspInit\(\)`](#)
- [`HAL\_CRC\_MspDeInit\(\)`](#)

### 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.

This section contains the following APIs:

- [\*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.

This section contains the following APIs:

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

### 8.2.5 HAL\_CRC\_Init

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

### 8.2.6 HAL\_CRC\_DeInit

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

### 8.2.7 HAL\_CRC\_MspInit

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

### 8.2.8 HAL\_CRC\_MspDeInit

Function Name	<code>void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)</code>
---------------	--

	Function Description	Deinitializes the CRC MSP.
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>• None</li> </ul>
<b>8.2.9 HAL_CRC_Accumulate</b>		
Function Name	<b>uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * 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 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>• 32-bit CRC</li> </ul>
<b>8.2.10 HAL_CRC_Calculate</b>		
Function Name	<b>uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * 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>• 32-bit CRC</li> </ul>
<b>8.2.11 HAL_CRC_GetState</b>		
Function Name	<b>HAL_CRC_StateTypeDef HAL_CRC_GetState (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>• HAL state</li> </ul>

## 8.3 CRC Firmware driver defines

### 8.3.1 CRC

#### *CRC Exported Macros*

**\_HAL\_CRC\_RESET\_HANDLE\_STATE    Description:**

- Resets CRC handle state.

**Parameters:**

- `__HANDLE__`: CRC handle

**Return value:**

- None

`__HAL_CRC_DR_RESET`

**Description:**

- Resets CRC Data Register.

**Parameters:**

- `__HANDLE__`: CRC handle

**Return value:**

- None

`__HAL_CRC_SET_IDR`

**Description:**

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

**Parameters:**

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

**Return value:**

- None

`__HAL_CRC_GET_IDR`

**Description:**

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

**Parameters:**

- `__HANDLE__`: CRC handle

**Return value:**

- 8-bit: value of the ID register

## 9 HAL CRYP Generic Driver

### 9.1 CRYP Firmware driver registers structures

#### 9.1.1 CRYP\_InitTypeDef

##### Data Fields

- *uint32\_t DataType*
- *uint32\_t KeySize*
- *uint8\_t \* pKey*
- *uint8\_t \* pInitVect*
- *uint8\_t IVSize*
- *uint8\_t TagSize*
- *uint8\_t \* Header*
- *uint32\_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
- ***uint32\_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

#### 9.1.2 CRYP\_HandleTypeDefDef

##### Data Fields



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

## 9.2 CRYP Firmware driver API description

### 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 `_HAL_RCC_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. It's 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\_DelInit() 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

This section contains the following APIs:

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

## 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

This section contains the following APIs:

- [\*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\\_CRYP\\_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

This section contains the following APIs:

- [\*HAL\\_CRYP\\_DESECB\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYP\\_DESECB\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYP\\_DESCBC\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYP\\_DESCBC\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYP\\_DESECB\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYP\\_DESCBC\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYP\\_DESECB\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYP\\_DESCBC\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYP\\_DESECB\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYP\\_DESCBC\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYP\\_DESECB\\_Decrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYP\\_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

This section contains the following APIs:

- `HAL_CRYP_TDESECB_Encrypt()`
- `HAL_CRYP_TDESECB_Decrypt()`
- `HAL_CRYP_TDESCBC_Encrypt()`
- `HAL_CRYP_TDESCBC_Decrypt()`
- `HAL_CRYP_TDESECB_Encrypt_IT()`
- `HAL_CRYP_TDESCBC_Encrypt_IT()`
- `HAL_CRYP_TDESECB_Decrypt_IT()`
- `HAL_CRYP_TDESCBC_Decrypt_IT()`
- `HAL_CRYP_TDESECB_Encrypt_DMA()`
- `HAL_CRYP_TDESCBC_Encrypt_DMA()`
- `HAL_CRYP_TDESECB_Decrypt_DMA()`
- `HAL_CRYP_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

This section contains the following APIs:

- `HAL_CRYP_InCpltCallback()`
- `HAL_CRYP_OutCpltCallback()`
- `HAL_CRYP_ErrorCallback()`

## 9.2.7 CRYP IRQ handler management

This section provides CRYP IRQ handler function.

This section contains the following APIs:

- `HAL_CRYP_IRQHandler()`

## 9.2.8 Peripheral State functions

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

This section contains the following APIs:

- `HAL_CRYP_GetState()`

## 9.2.9 HAL\_CRYP\_Init

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef *hcryp)</code>
---------------	--

Function Description	Initializes the CRYP according to the specified parameters in the CRYP_InitTypeDef and creates the associated handle.
----------------------	---

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>• HAL status</li> </ul>

### 9.2.10 HAL\_CRYP\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DelInit (CRYP_HandleTypeDef * hcryp)</b>
Function Description	Deinitializes the CRYP peripheral.
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>• HAL status</li> </ul>

### 9.2.11 HAL\_CRYP\_MspInit

Function Name	<b>void HAL_CRYP_MspInit (CRYP_HandleTypeDef * 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>

### 9.2.12 HAL\_CRYP\_MspDelInit

Function Name	<b>void HAL_CRYP_MspDelInit (CRYP_HandleTypeDef * 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>

### 9.2.13 HAL\_CRYP\_AESECB\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt (CRYP_HandleTypeDef * 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>• HAL status</li> </ul>

### 9.2.14 HAL\_CRYP\_AESCBC\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt</b>
---------------	--

**(CRYP\_HandleTypeDef \* hcryp, uint8\_t \* pPlainData, uint16\_t Size, uint8\_t \* pCypherData, uint32\_t Timeout)**

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>• HAL status</li> </ul>

### 9.2.15 HAL\_CRYP\_AESCTR\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt</b> <b>(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</b>
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> <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>• HAL status</li> </ul>

### 9.2.16 HAL\_CRYP\_AESECB\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt</b> <b>(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</b>
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>• HAL status</li> </ul>

### 9.2.17 HAL\_CRYP\_AESCBC\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt</b> <b>(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</b>
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>HAL status</li> </ul>

### 9.2.18 HAL\_CRYP\_AESCTR\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, 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>HAL status</li> </ul>

### 9.2.19 HAL\_CRYP\_AESECB\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t 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>HAL status</li> </ul>

### 9.2.20 HAL\_CRYP\_AESCBC\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
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> </ul>

- **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes
- **pCypherData:** Pointer to the ciphertext buffer
- HAL status

Return values

### 9.2.21 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>• HAL status</li> </ul>

### 9.2.22 HAL\_CRYP\_AESECB\_Decrypt\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_IT (CRYP_HandleTypeDef * 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>• HAL status</li> </ul>

### 9.2.23 HAL\_CRYP\_AESCBC\_Decrypt\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_IT (CRYP_HandleTypeDef * 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>• HAL status</li> </ul>

**9.2.24 HAL\_CRYP\_AESCTR\_Decrypt\_IT**

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</code>
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>• HAL status</li> </ul>

**9.2.25 HAL\_CRYP\_AESECB\_Encrypt\_DMA**

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESECB_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</code>
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>• HAL status</li> </ul>

**9.2.26 HAL\_CRYP\_AESCBC\_Encrypt\_DMA**

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_DMA (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 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**9.2.27 HAL\_CRYP\_AESCTR\_Encrypt\_DMA**

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_DMA (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

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.</li> <li>• <b>pPlainData:</b> Pointer to the plaintext buffer</li> </ul> |
| Return values | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>  |

### 9.2.28 HAL\_CRYP\_AESECB\_Decrypt\_DMA

- |                      |  |
|----------------------|--|
| Function Name        | <code>HAL_StatusTypeDef HAL_CRYP_AESECB_Decrypt_DMA(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, 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>• HAL status</li> </ul>   |

### 9.2.29 HAL\_CRYP\_AESCBC\_Decrypt\_DMA

- |                      |  |
|----------------------|--|
| Function Name        | <code>HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt_DMA(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</code>   |
| 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>• HAL status</li> </ul>   |

### 9.2.30 HAL\_CRYP\_AESCTR\_Decrypt\_DMA

- |                      |  |
|----------------------|--|
| Function Name        | <code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Decrypt_DMA(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</code>   |
| 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</li> </ul> |

- **pPlainData:** Pointer to the plaintext buffer

Return values • HAL status

### 9.2.31 HAL\_CRYP\_DESECB\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</b>
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	• HAL status

### 9.2.32 HAL\_CRYP\_DESECB\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t 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	• HAL status

### 9.2.33 HAL\_CRYP\_DESCBC\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t 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	• HAL status

### 9.2.34 HAL\_CRYP\_DESCBC\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt (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>HAL status</li> </ul>
<b>9.2.35</b>	<b>HAL_CRYP_DESECB_Encrypt_IT</b>	
Function Name		<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t 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>HAL status</li> </ul>
<b>9.2.36</b>	<b>HAL_CRYP_DESCBC_Encrypt_IT</b>	
Function Name		<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t 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>HAL status</li> </ul>
<b>9.2.37</b>	<b>HAL_CRYP_DESECB_Decrypt_IT</b>	
Function Name		<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, 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> </ul>

- **Size:** Length of the plaintext buffer, must be a multiple of 8
- **pCypherData:** Pointer to the ciphertext buffer
- HAL status

Return values

**9.2.38 HAL\_CRYP\_DESCBC\_Decrypt\_IT**

Function Name

**HAL\_StatusTypeDef HAL\_CRYP\_DESCBC\_Decrypt\_IT  
(CRYP\_HandleTypeDef \* hcryp, uint8\_t \* pCypherData,  
uint16\_t Size, uint8\_t \* pPlainData)**

Function Description

Initializes the CRYP peripheral in DES ECB decryption mode using interrupt.

Parameters

- **hcryp:** pointer to a CRYP\_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer
- **Size:** Length of the plaintext buffer, must be a multiple of 8
- **pCypherData:** Pointer to the ciphertext buffer

Return values

- HAL status

**9.2.39 HAL\_CRYP\_DESECB\_Encrypt\_DMA**

Function Name

**HAL\_StatusTypeDef HAL\_CRYP\_DESECB\_Encrypt\_DMA  
(CRYP\_HandleTypeDef \* hcryp, uint8\_t \* pPlainData, uint16\_t  
Size, uint8\_t \* pCypherData)**

Function Description

Initializes the CRYP peripheral in DES ECB encryption mode using DMA.

Parameters

- **hcryp:** pointer to a CRYP\_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer
- **Size:** Length of the plaintext buffer, must be a multiple of 8
- **pCypherData:** Pointer to the ciphertext buffer

Return values

- HAL status

**9.2.40 HAL\_CRYP\_DESCBC\_Encrypt\_DMA**

Function Name

**HAL\_StatusTypeDef HAL\_CRYP\_DESCBC\_Encrypt\_DMA  
(CRYP\_HandleTypeDef \* hcryp, uint8\_t \* pPlainData, uint16\_t  
Size, uint8\_t \* pCypherData)**

Function Description

Initializes the CRYP peripheral in DES CBC encryption mode using DMA.

Parameters

- **hcryp:** pointer to a CRYP\_HandleTypeDef structure that contains the configuration information for CRYP module
- **pPlainData:** Pointer to the plaintext buffer
- **Size:** Length of the plaintext buffer, must be a multiple of 8
- **pCypherData:** Pointer to the ciphertext buffer

Return values

- HAL status

**9.2.41 HAL\_CRYP\_DESECB\_Decrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESECB_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 9.2.42 HAL\_CRYP\_DESCBC\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 9.2.43 HAL\_CRYP\_TDESECB\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</b>
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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 9.2.44 HAL\_CRYP\_TDESECB\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</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	HAL status

#### 9.2.45 HAL\_CRYP\_TDESCBC\_Encrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_TDESCBC_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 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	HAL status

#### 9.2.46 HAL\_CRYP\_TDESCBC\_Decrypt

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</code>
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	HAL status

#### 9.2.47 HAL\_CRYP\_TDESECB\_Encrypt\_IT

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</code>
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> </ul>

	<ul style="list-style-type: none"> <li><b>pCypherData:</b> Pointer to the ciphertext buffer</li> <li>HAL status</li> </ul>
Return values	
<b>9.2.48 HAL_CRYP_TDESCBC_Encrypt_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t 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>HAL status</li> </ul>
<b>9.2.49 HAL_CRYP_TDESECB_Decrypt_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, 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>HAL status</li> </ul>
<b>9.2.50 HAL_CRYP_TDESCBC_Decrypt_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, 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>HAL status</li> </ul>
<b>9.2.51 HAL_CRYP_TDESECB_Encrypt_DMA</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYP_TDESECB_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
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>• HAL status</li> </ul>  |

### 9.2.52 HAL\_CRYP\_TDESCBC\_Encrypt\_DMA

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Encrypt_DMA(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>   |
| 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>• HAL status</li> </ul>  |

### 9.2.53 HAL\_CRYP\_TDESECB\_Decrypt\_DMA

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYP_TDESECB_Decrypt_DMA(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</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>• HAL status</li> </ul>  |

### 9.2.54 HAL\_CRYP\_TDESCBC\_Decrypt\_DMA

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYP_TDESCBC_Decrypt_DMA(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</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>• HAL status</li> </ul>
<b>9.2.55</b>	<b>HAL_CRYP_InCpltCallback</b>	
Function Name	<b>void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)</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>	
<b>9.2.56</b>	<b>HAL_CRYP_OutCpltCallback</b>	
Function Name	<b>void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)</b>	
Function Description	Output 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>	
<b>9.2.57</b>	<b>HAL_CRYP_ErrorCallback</b>	
Function Name	<b>void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)</b>	
Function Description	CRYP error 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>	
<b>9.2.58</b>	<b>HAL_CRYP_IRQHandler</b>	
Function Name	<b>void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * 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>	
<b>9.2.59</b>	<b>HAL_CRYP_GetState</b>	
Function Name	<b>HAL_CRYP_STATETypeDef HAL_CRYP_GetState (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>• HAL state</li> </ul>	

## 9.3 CRYP Firmware driver defines

### 9.3.1 CRYP

#### *CRYP\_Key\_Size*

CRYP\_KEYSIZE\_128B

CRYP\_KEYSIZE\_192B

CRYP\_KEYSIZE\_256B

#### *CRYP\_Data\_Type*

CRYP\_DATATYPE\_32B

CRYP\_DATATYPE\_16B

CRYP\_DATATYPE\_8B

CRYP\_DATATYPE\_1B

#### *CRYP\_CRYP\_AlgoModeDirection*

CRYP\_CR\_ALGOMODE\_DIRECTION

CRYP\_CR\_ALGOMODE\_TDES\_ECB\_ENCRYPT

CRYP\_CR\_ALGOMODE\_TDES\_ECB\_DECRYPT

CRYP\_CR\_ALGOMODE\_TDES\_CBC\_ENCRYPT

CRYP\_CR\_ALGOMODE\_TDES\_CBC\_DECRYPT

CRYP\_CR\_ALGOMODE DES\_ECB\_ENCRYPT

CRYP\_CR\_ALGOMODE DES\_ECB\_DECRYPT

CRYP\_CR\_ALGOMODE DES\_CBC\_ENCRYPT

CRYP\_CR\_ALGOMODE DES\_CBC\_DECRYPT

CRYP\_CR\_ALGOMODE AES\_ECB\_ENCRYPT

CRYP\_CR\_ALGOMODE AES\_ECB\_DECRYPT

CRYP\_CR\_ALGOMODE AES\_CBC\_ENCRYPT

CRYP\_CR\_ALGOMODE AES\_CBC\_DECRYPT

CRYP\_CR\_ALGOMODE AES\_CTR\_ENCRYPT

CRYP\_CR\_ALGOMODE AES\_CTR\_DECRYPT

#### *CRYP\_CRYP\_Interrupt*

CRYP\_IT\_INI Input FIFO Interrupt

CRYP\_IT\_OUTI Output FIFO Interrupt

#### *CRYP\_CRYP\_Flags*

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

CRYP\_FLAG\_IFEM Input FIFO is empty

CRYP\_FLAG\_IFNF Input FIFO is not Full

CRYP\_FLAG\_OFNE Output FIFO is not empty

CRYP_FLAG_OFFU	Output FIFO is Full
CRYP_FLAG_OUTRIS	Output FIFO service raw interrupt status
CRYP_FLAG_INRIS	Input FIFO service raw interrupt status

***CRYP Exported Macros*****\_HAL\_CRYP\_RESET\_HANDLE\_STATE** **Description:**

- Reset CRYP handle state.

**Parameters:**

- HANDLE: specifies the CRYP handle.

**Return value:**

- None

**\_HAL\_CRYP\_ENABLE****Description:**

- Enable/Disable the CRYP peripheral.

**Parameters:**

- HANDLE: specifies the CRYP handle.

**Return value:**

- None

**\_HAL\_CRYP\_DISABLE****Description:**

- Flush the data FIFO.

**Parameters:**

- HANDLE: specifies the CRYP handle.

**Return value:**

- None

**\_HAL\_CRYP\_SET\_MODE****Description:**

- Set the algorithm mode: AES-ECB, AES-CBC, AES-CTR, DES-ECB, DES-CBC.

**Parameters:**

- HANDLE: specifies the CRYP handle.
- MODE: The algorithm mode.

**Return value:**

- None

**\_HAL\_CRYP\_GET\_FLAG****Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the CRYP handle.
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - CRYP\_FLAG\_BUSY: The CRYP core is currently processing a block of data or a key preparation (for AES decryption).
  - CRYP\_FLAG\_IFEM: Input FIFO is empty
  - CRYP\_FLAG\_IFNF: Input FIFO is not full
  - CRYP\_FLAG\_INRIS: Input FIFO service raw interrupt is pending
  - CRYP\_FLAG\_OFNE: Output FIFO is not empty
  - CRYP\_FLAG\_OFFU: Output FIFO is full
  - CRYP\_FLAG\_OUTRIS: Input FIFO service raw interrupt is pending

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

[\\_\\_HAL\\_CRYP\\_GET\\_IT](#)**Description:**

- Check whether the specified CRYP interrupt is set or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the CRYP handle.
- \_\_INTERRUPT\_\_: specifies the interrupt to check. This parameter can be one of the following values:
  - CRYP\_IT\_INRIS: Input FIFO service raw interrupt is pending
  - CRYP\_IT\_OUTRIS: Output FIFO service raw interrupt is pending

**Return value:**

- The: new state of \_\_INTERRUPT\_\_ (TRUE or FALSE).

[\\_\\_HAL\\_CRYP\\_ENABLE\\_IT](#)**Description:**

- Enable the CRYP interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the CRYP handle.
- \_\_INTERRUPT\_\_: CRYP Interrupt.

**Return value:**

- None

`__HAL_CRYP_DISABLE_IT`

**Description:**

- Disable the CRYP interrupt.

**Parameters:**

- `__HANDLE__`: specifies the CRYP handle.
- `__INTERRUPT__`: CRYP interrupt.

**Return value:**

- None

***CRYP Private Constants***

`CRYP_FLAG_MASK`

***CRYP Private define***

`CRYP_TIMEOUT_VALUE`

***CRYP Private Macros***

`IS_CRYP_KEYSIZE`

`IS_CRYP_DATATYPE`

## 10 HAL DAC Generic Driver

### 10.1 DAC Firmware driver registers structures

#### 10.1.1 DAC\_HandleTypeDef

##### Data Fields

- *DAC\_TypeDef \* Instance*
- *\_\_IO HAL\_DAC\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*
- *DMA\_HandleTypeDef \* DMA\_Handle1*
- *DMA\_HandleTypeDef \* DMA\_Handle2*
- *\_\_IO uint32\_t ErrorCode*

##### Field Documentation

- ***DAC\_TypeDef\* DAC\_HandleTypeDef::Instance***  
Register base address
- ***\_\_IO HAL\_DAC\_StateTypeDef DAC\_HandleTypeDef::State***  
DAC communication state
- ***HAL\_LockTypeDef DAC\_HandleTypeDef::Lock***  
DAC locking object
- ***DMA\_HandleTypeDef\* DAC\_HandleTypeDef::DMA\_Handle1***  
Pointer DMA handler for channel 1
- ***DMA\_HandleTypeDef\* DAC\_HandleTypeDef::DMA\_Handle2***  
Pointer DMA handler for channel 2
- ***\_\_IO uint32\_t DAC\_HandleTypeDef::ErrorCode***  
DAC Error code

#### 10.1.2 DAC\_ChannelConfTypeDef

##### 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\\_buffer\*\*](#)

## 10.2 DAC Firmware driver API description

### 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, 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.

This section contains the following APIs:

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

### 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.

This section contains the following APIs:

- [\*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\\_DMADebugCallbackCh1\(\)\*](#)

### 10.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- [\*HAL\\_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.

This section contains the following APIs:

- `HAL_DAC_GetState()`
- `HAL_DAC_GetError()`
- `HAL_DAC_IRQHandler()`
- `HAL_DAC_ConvCpltCallbackCh1()`
- `HAL_DAC_ConvHalfCpltCallbackCh1()`
- `HAL_DAC_ErrorCallbackCh1()`
- `HAL_DAC_DMAUnderrunCallbackCh1()`

### 10.2.7 HAL\_DAC\_Init

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

### 10.2.8 HAL\_DAC\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)</code>
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 <code>DAC_HandleTypeDef</code> structure that contains the configuration information for the specified DAC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 10.2.9 HAL\_DAC\_MspInit

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

### 10.2.10 HAL\_DAC\_MspDeInit

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

### 10.2.11 HAL\_DAC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef *hdac, uint32_t 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 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 Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 10.2.12 HAL\_DAC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef *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 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 Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 10.2.13 HAL\_DAC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * 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 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 Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected</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: DAC_ALIGN_8B_R: 8bit right data alignment selectedDAC_ALIGN_12B_L: 12bit left data alignment selectedDAC_ALIGN_12B_R: 12bit right data alignment selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 10.2.14 HAL\_DAC\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * 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 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 Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 10.2.15 HAL\_DAC\_GetValue

Function Name	<b>uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * 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 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 Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>The selected DAC channel data output value.</li> </ul>

#### 10.2.16 HAL\_DAC\_IRQHandler

Function Name	<b>void HAL_DAC_IRQHandler (DAC_HandleTypeDef * 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>

#### 10.2.17 HAL\_DAC\_ConvCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * 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>

#### 10.2.18 HAL\_DAC\_ConvHalfCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * 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>

### 10.2.19 HAL\_DAC\_ErrorCallbackCh1

Function Name	<code>void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)</code>
Function Description	Error DAC callback 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>

### 10.2.20 HAL\_DAC\_DMADebugCallbackCh1

Function Name	<code>void HAL_DAC_DMADebugCallbackCh1 (DAC_HandleTypeDef * hdac)</code>
Function Description	DMA debug DAC callback 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>

### 10.2.21 HAL\_DAC\_ConfigChannel

Function Name	<code>HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)</code>
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: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 10.2.22 HAL\_DAC\_SetValue

Function Name	<code>HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)</code>
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: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2</li> </ul>

selected	<ul style="list-style-type: none"> <li>• <b>Alignment:</b> Specifies the data alignment. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selectedDAC_ALIGN_12B_L: 12bit left data alignment selectedDAC_ALIGN_12B_R: 12bit right data alignment selected</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>• HAL status</li> </ul>

### 10.2.23 HAL\_DAC\_GetState

Function Name	<b>HAL_DAC_StateTypeDef HAL_DAC_GetState (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>• HAL state</li> </ul>

### 10.2.24 HAL\_DAC\_GetError

Function Name	<b>uint32_t HAL_DAC_GetError (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>• DAC Error Code</li> </ul>

### 10.2.25 HAL\_DAC\_IRQHandler

Function Name	<b>void HAL_DAC_IRQHandler (DAC_HandleTypeDef * 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>

### 10.2.26 HAL\_DAC\_ConvCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * 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>

### 10.2.27 HAL\_DAC\_ConvHalfCpltCallbackCh1

Function Name	<b>void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * 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>

### 10.2.28 HAL\_DAC\_ErrorCallbackCh1

Function Name	<b>void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)</b>
Function Description	Error DAC callback 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>

### 10.2.29 HAL\_DAC\_DMAUnderrunCallbackCh1

Function Name	<b>void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)</b>
Function Description	DMA underrun DAC callback 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>

## 10.3 DAC Firmware driver defines

### 10.3.1 DAC

#### *DAC Channel Selection*

DAC\_CHANNEL\_1

DAC\_CHANNEL\_2

#### *DAC Data Alignment*

DAC\_ALIGN\_12B\_R

DAC\_ALIGN\_12B\_L

DAC\_ALIGN\_8B\_R

#### *DAC Error Code*

HAL_DAC_ERROR_NONE	No error
--------------------	----------

HAL_DAC_ERROR_DMAUNDERUNCH1	DAC channel1 DAM underrun error
-----------------------------	---------------------------------

HAL_DAC_ERROR_DMAUNDERUNCH2	DAC channel2 DAM underrun error
-----------------------------	---------------------------------

HAL_DAC_ERROR_DMA	DMA error
-------------------	-----------

#### *DAC Exported Macros*

<u>__HAL_DAC_RESET_HANDLE_STATE</u>	<b>Description:</b>
-------------------------------------	---------------------

- Reset DAC handle state.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle.

**Return value:**

- None

**Description:**

- Enable the DAC channel.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle.
- `__DAC_Channel__`: specifies the DAC channel

**Return value:**

- None

**Description:**

- Disable the DAC channel.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle
- `__DAC_Channel__`: specifies the DAC channel.

**Return value:**

- None

**Description:**

- Enable the DAC interrupt.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt.

**Return value:**

- None

**Description:**

- Disable the DAC interrupt.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt.

**Return value:**

- None

**Description:**

- Checks if the specified DAC interrupt

source is enabled or disabled.

**Parameters:**

- `__HANDLE__`: DAC handle
- `__INTERRUPT__`: DAC interrupt source to check This parameter can be any combination of the following values:
  - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
  - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

**Return value:**

- State: of interruption (SET or RESET)

`__HAL_DAC_GET_FLAG`

**Description:**

- Get the selected DAC's flag status.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DAC_FLAG_DMAUDR1`: DMA underrun 1 flag
  - `DAC_FLAG_DMAUDR2`: DMA underrun 2 flag

**Return value:**

- None

`__HAL_DAC_CLEAR_FLAG`

**Description:**

- Clear the DAC's flag.

**Parameters:**

- `__HANDLE__`: specifies the DAC handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DAC_FLAG_DMAUDR1`: DMA underrun 1 flag
  - `DAC_FLAG_DMAUDR2`: DMA underrun 2 flag

**Return value:**

- None

**DAC Flags Definition**

`DAC_FLAG_DMAUDR1`

`DAC_FLAG_DMAUDR2`

**DAC IT Definition**

`DAC_IT_DMAUDR1`

DAC\_IT\_DMAUDR2

**DAC Output Buffer**

DAC\_OUTPUTBUFFER\_ENABLE

DAC\_OUTPUTBUFFER\_DISABLE

**DAC Private Macros**

IS\_DAC\_DATA

IS\_DAC\_ALIGN

IS\_DAC\_CHANNEL

IS\_DAC\_OUTPUT\_BUFFER\_STATE

IS\_DAC\_TRIGGER

DAC\_DHR12R1\_ALIGNMENT

**Description:**

- Set DHR12R1 alignment.

**Parameters:**

- ALIGNMENT: specifies the DAC alignment

**Return value:**

- None

DAC\_DHR12R2\_ALIGNMENT

**Description:**

- Set DHR12R2 alignment.

**Parameters:**

- ALIGNMENT: specifies the DAC alignment

**Return value:**

- None

DAC\_DHR12RD\_ALIGNMENT

**Description:**

- Set DHR12RD alignment.

**Parameters:**

- ALIGNMENT: specifies the DAC alignment

**Return value:**

- None

**DAC Trigger Selection**

DAC\_TRIGGER\_NONE

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

DAC\_TRIGGER\_T2\_TRGO

TIM2 TRGO selected as external conversion trigger for DAC channel

DAC\_TRIGGER\_T4\_TRGO

TIM4 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T5_TRGO	TIM5 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T8_TRGO	TIM8 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_EXT_IT9	EXTI Line9 event selected as external conversion trigger for DAC channel
DAC_TRIGGER_SOFTWARE	Conversion started by software trigger for DAC channel

## 11 HAL DAC Extension Driver

### 11.1 DACEx Firmware driver API description

#### 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.

This section contains the following APIs:

- [\*HAL\\_DACEx\\_DualGetValue\(\)\*](#)
- [\*HAL\\_DACEx\\_TriangleWaveGenerate\(\)\*](#)
- [\*HAL\\_DACEx\\_NoiseWaveGenerate\(\)\*](#)
- [\*HAL\\_DACEx\\_DualSetValue\(\)\*](#)
- [\*HAL\\_DACEx\\_ConvCpltCallbackCh2\(\)\*](#)
- [\*HAL\\_DACEx\\_ConvHalfCpltCallbackCh2\(\)\*](#)
- [\*HAL\\_DACEx\\_ErrorCallbackCh2\(\)\*](#)
- [\*HAL\\_DACEx\\_DMAUnderrunCallbackCh2\(\)\*](#)

#### 11.1.3 HAL\_DACEx\_DualGetValue

Function Name	<code>uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef *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>• The selected DAC channel data output value.</li></ul>

#### 11.1.4 HAL\_DACEx\_TriangleWaveGenerate

Function Name	<code>HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef *hdac, uint32_t Channel, uint32_t 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: DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 11.1.5 HAL\_DACEx\_NoiseWaveGenerate

Function Name	<b>HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)</b>
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: DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generationDAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generationDAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generationDAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generationDAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generationDAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generationDAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave</li> </ul>

generationDAC\_LFSRUNMASK\_BITS7\_0: Unmask DAC channel LFSR bit[7:0] for noise wave  
 generationDAC\_LFSRUNMASK\_BITS8\_0: Unmask DAC channel LFSR bit[8:0] for noise wave  
 generationDAC\_LFSRUNMASK\_BITS9\_0: Unmask DAC channel LFSR bit[9:0] for noise wave  
 generationDAC\_LFSRUNMASK\_BITS10\_0: Unmask DAC channel LFSR bit[10:0] for noise wave  
 generationDAC\_LFSRUNMASK\_BITS11\_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

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

### 11.1.6 HAL\_DACEx\_DualSetValue

Function Name	<b>HAL_StatusTypeDef HAL_DACEx_DualSetValue</b> <b>(DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t 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>• HAL status</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>

### 11.1.7 HAL\_DACEx\_ConvCpltCallbackCh2

Function Name	<b>void HAL_DACEx_ConvCpltCallbackCh2</b> <b>(DAC_HandleTypeDef * hdac)</b>
Function Description	Conversion complete callback in non blocking mode for Channel2.
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>

### 11.1.8 HAL\_DACEx\_ConvHalfCpltCallbackCh2

Function Name	<b>void HAL_DACEx_ConvHalfCpltCallbackCh2</b> <b>(DAC_HandleTypeDef * hdac)</b>
Function Description	Conversion half DMA transfer callback in non blocking mode for Channel2.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdac:</b> pointer to a DAC_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified DAC.

Return values • None

### 11.1.9 HAL\_DACEx\_ErrorCallbackCh2

Function Name **void HAL\_DACEx\_ErrorCallbackCh2 (DAC\_HandleTypeDef \* hdac)**

Function Description Error DAC callback for Channel2.

Parameters • **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values • None

### 11.1.10 HAL\_DACEx\_DMAUnderrunCallbackCh2

Function Name **void HAL\_DACEx\_DMAUnderrunCallbackCh2 (DAC\_HandleTypeDef \* hdac)**

Function Description DMA underrun DAC callback for channel2.

Parameters • **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values • None

## 11.2 DACEx Firmware driver defines

### 11.2.1 DACEx

#### *DAC LFS Run Mask Triangle Amplitude*

DAC_LFSRUNMASK_BIT0	Unmask DAC channel LFSR bit0 for noise wave generation
DAC_LFSRUNMASK_BITS1_0	Unmask DAC channel LFSR bit[1:0] for noise wave generation
DAC_LFSRUNMASK_BITS2_0	Unmask DAC channel LFSR bit[2:0] for noise wave generation
DAC_LFSRUNMASK_BITS3_0	Unmask DAC channel LFSR bit[3:0] for noise wave generation
DAC_LFSRUNMASK_BITS4_0	Unmask DAC channel LFSR bit[4:0] for noise wave generation
DAC_LFSRUNMASK_BITS5_0	Unmask DAC channel LFSR bit[5:0] for noise wave generation
DAC_LFSRUNMASK_BITS6_0	Unmask DAC channel LFSR bit[6:0] for noise wave generation
DAC_LFSRUNMASK_BITS7_0	Unmask DAC channel LFSR bit[7:0] for noise wave generation
DAC_LFSRUNMASK_BITS8_0	Unmask DAC channel LFSR bit[8:0] for noise wave generation
DAC_LFSRUNMASK_BITS9_0	Unmask DAC channel LFSR bit[9:0] for noise wave

	generation
DAC_LFSRUNMASK_BITS10_0	Unmask DAC channel LFSR bit[10:0] for noise wave generation
DAC_LFSRUNMASK_BITS11_0	Unmask DAC channel LFSR bit[11:0] for noise wave generation
DAC_TRIANGLEAMPLITUDE_1	Select max triangle amplitude of 1
DAC_TRIANGLEAMPLITUDE_3	Select max triangle amplitude of 3
DAC_TRIANGLEAMPLITUDE_7	Select max triangle amplitude of 7
DAC_TRIANGLEAMPLITUDE_15	Select max triangle amplitude of 15
DAC_TRIANGLEAMPLITUDE_31	Select max triangle amplitude of 31
DAC_TRIANGLEAMPLITUDE_63	Select max triangle amplitude of 63
DAC_TRIANGLEAMPLITUDE_127	Select max triangle amplitude of 127
DAC_TRIANGLEAMPLITUDE_255	Select max triangle amplitude of 255
DAC_TRIANGLEAMPLITUDE_511	Select max triangle amplitude of 511
DAC_TRIANGLEAMPLITUDE_1023	Select max triangle amplitude of 1023
DAC_TRIANGLEAMPLITUDE_2047	Select max triangle amplitude of 2047
DAC_TRIANGLEAMPLITUDE_4095	Select max triangle amplitude of 4095

***DAC Private Macros***`IS_DAC_LFSR_UNMASK_TRIANGLE_AMPLITUDE`

## 12 HAL DCMI Generic Driver

### 12.1 DCMI Firmware driver registers structures

#### 12.1.1 DCMI\_CodesInitTypeDef

##### Data Fields

- *uint8\_t FrameStartCode*
- *uint8\_t LineStartCode*
- *uint8\_t LineEndCode*
- *uint8\_t FrameEndCode*

##### Field Documentation

- *uint8\_t DCMI\_CodesInitTypeDef::FrameStartCode*  
Specifies the code of the frame start delimiter.
- *uint8\_t DCMI\_CodesInitTypeDef::LineStartCode*  
Specifies the code of the line start delimiter.
- *uint8\_t DCMI\_CodesInitTypeDef::LineEndCode*  
Specifies the code of the line end delimiter.
- *uint8\_t DCMI\_CodesInitTypeDef::FrameEndCode*  
Specifies the code of the frame end delimiter.

#### 12.1.2 DCMI\_InitTypeDef

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

### 12.1.3 DCMI\_HandleTypeDef

#### Data Fields

- ***DCMI\_TypeDef \* Instance***
- ***DCMI\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_IO HAL\_DCMI\_StateTypeDef State***
- ***\_IO uint32\_t XferCount***
- ***\_IO uint32\_t XferSize***
- ***uint32\_t XferTransferNumber***
- ***uint32\_t pBuffPtr***
- ***DMA\_HandleTypeDef \* DMA\_Handle***
- ***\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***DCMI\_TypeDef\* DCMI\_HandleTypeDef::Instance***  
DCMI Register base address
- ***DCMI\_InitTypeDef DCMI\_HandleTypeDef::Init***  
DCMI parameters
- ***HAL\_LockTypeDef DCMI\_HandleTypeDef::Lock***  
DCMI locking object
- ***\_IO HAL\_DCMI\_StateTypeDef DCMI\_HandleTypeDef::State***  
DCMI state
- ***\_IO uint32\_t DCMI\_HandleTypeDef::XferCount***  
DMA transfer counter
- ***\_IO uint32\_t DCMI\_HandleTypeDef::XferSize***  
DMA transfer size
- ***uint32\_t DCMI\_HandleTypeDef::XferTransferNumber***  
DMA transfer number
- ***uint32\_t DCMI\_HandleTypeDef::pBuffPtr***  
Pointer to DMA output buffer

- **DMA\_HandleTypeDef\* DCMI\_HandleTypeDef::DMA\_Handle**  
Pointer to the DMA handler
- **\_IO uint32\_t DCMI\_HandleTypeDef::ErrorCode**  
DCMI Error code

## 12.2 DCMI Firmware driver API description

### 12.2.1 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

### 12.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI

This section contains the following APIs:

- [\*\*HAL\\_DCMI\\_Init\(\)\*\*](#)

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

### 12.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length and Enables DCMI DMA request and enables DCMI capture
- Stop the DCMI capture.
- Handles DCMI interrupt request.

This section contains the following APIs:

- [\*HAL\\_DCMI\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_DCMI\\_Stop\(\)\*](#)
- [\*HAL\\_DCMI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_DCMI\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_LineEventCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_VsyncEventCallback\(\)\*](#)
- [\*HAL\\_DCMI\\_FrameEventCallback\(\)\*](#)

### 12.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the CROP feature.
- Enable/Disable the CROP feature.

This section contains the following APIs:

- [\*HAL\\_DCMI\\_ConfigCROP\(\)\*](#)
- [\*HAL\\_DCMI\\_DisableCROP\(\)\*](#)
- [\*HAL\\_DCMI\\_EnableCROP\(\)\*](#)

### 12.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DCMI state.
- Get the specific DCMI error flag.

This section contains the following APIs:

- [\*HAL\\_DCMI\\_GetState\(\)\*](#)
- [\*HAL\\_DCMI\\_GetError\(\)\*](#)

### 12.2.6 HAL\_DCMI\_Init

Function Name	<code>HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef *hdcmi)</code>
Function Description	Initializes the DCMI according to the specified parameters in the <code>DCMI_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi:</b> pointer to a <code>DCMI_HandleTypeDef</code> structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 12.2.7 HAL\_DCMI\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_DelInit (DCMI_HandleTypeDefDef * hdcmi)</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_HandleTypeDefDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 12.2.8 HAL\_DCMI\_MspInit

Function Name	<b>void HAL_DCMI_MspInit (DCMI_HandleTypeDefDef * hdcmi)</b>
Function Description	Initializes the DCMI MSP.
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>

### 12.2.9 HAL\_DCMI\_MspDelInit

Function Name	<b>void HAL_DCMI_MspDelInit (DCMI_HandleTypeDefDef * hdcmi)</b>
Function Description	Deinitializes the DCMI MSP.
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>

### 12.2.10 HAL\_DCMI\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_Start_DMA (DCMI_HandleTypeDefDef * 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_HandleTypeDefDef 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>• HAL status</li> </ul>

### 12.2.11 HAL\_DCMI\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_Stop (DCMI_HandleTypeDefDef * 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_HandleTypeDefDef structure that</li> </ul>

contains the configuration information for DCMI.

Return values

- HAL status

### 12.2.12 HAL\_DCMI\_IRQHandler

Function Name **void HAL\_DCMI\_IRQHandler (DCMI\_HandleTypeDef \* hdcmi)**

Function Description Handles DCMI interrupt request.

Parameters

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for the DCMI.

Return values

- None

### 12.2.13 HAL\_DCMI\_ErrorCallback

Function Name **void HAL\_DCMI\_ErrorCallback (DCMI\_HandleTypeDef \* hdcmi)**

Function Description Error DCMI callback.

Parameters

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- None

### 12.2.14 HAL\_DCMI\_LineEventCallback

Function Name **void HAL\_DCMI\_LineEventCallback (DCMI\_HandleTypeDef \* hdcmi)**

Function Description Line Event callback.

Parameters

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- None

### 12.2.15 HAL\_DCMI\_VsyncEventCallback

Function Name **void HAL\_DCMI\_VsyncEventCallback (DCMI\_HandleTypeDef \* hdcmi)**

Function Description VSYNC Event callback.

Parameters

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- None

### 12.2.16 HAL\_DCMI\_FrameEventCallback

Function Name **void HAL\_DCMI\_FrameEventCallback (DCMI\_HandleTypeDef \* hdcmi)**

Function Description Frame Event callback.

Parameters

- **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.

Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
<b>12.2.17 HAL_DCMI_ConfigCROP</b>	
Function Name	<b>HAL_StatusTypeDef HAL_DCMI_ConfigCROP (DCMI_HandleTypeDef * hdcmi, uint32_t X0, uint32_t Y0, uint32_t XSize, uint32_t YSize)</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>X0:</b> DCMI window X offset</li> <li><b>Y0:</b> DCMI window Y offset</li> <li><b>XSize:</b> DCMI Pixel per line</li> <li><b>YSize:</b> DCMI Line number</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
<b>12.2.18 HAL_DCMI_DisableCROP</b>	
Function Name	<b>HAL_StatusTypeDef HAL_DCMI_DisableCROP (DCMI_HandleTypeDef * hdcmi)</b>
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>HAL status</li> </ul>
<b>12.2.19 HAL_DCMI_EnableCROP</b>	
Function Name	<b>HAL_StatusTypeDef HAL_DCMI_EnableCROP (DCMI_HandleTypeDef * hdcmi)</b>
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>HAL status</li> </ul>
<b>12.2.20 HAL_DCMI_GetState</b>	
Function Name	<b>HAL_DCMI_StateTypeDef HAL_DCMI_GetState (DCMI_HandleTypeDef * 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>HAL state</li> </ul>
<b>12.2.21 HAL_DCMI_GetError</b>	
Function Name	<b>uint32_t HAL_DCMI_GetError (DCMI_HandleTypeDef * 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>• DCMI Error Code</li> </ul>

## 12.3 DCMI Firmware driver defines

### 12.3.1 DCMI

#### ***DCMI Capture Mode***

DCMI\_MODE\_CONTINUOUS The received data are transferred continuously into the destination memory through the DMA

DCMI\_MODE\_SNAPSHOT Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA

#### ***DCMI Capture Rate***

DCMI\_CR\_ALL\_FRAME All frames are captured

DCMI\_CR\_ALTERNATE\_2\_FRAME Every alternate frame captured

DCMI\_CR\_ALTERNATE\_4\_FRAME One frame in 4 frames captured

#### ***DCMI Error Code***

HAL\_DCMI\_ERROR\_NONE No error

HAL\_DCMI\_ERROR\_OVF Overflow error

HAL\_DCMI\_ERROR\_SYNC Synchronization error

HAL\_DCMI\_ERROR\_TIMEOUT Timeout error

#### ***DCMI Exported Macros***

**\_\_HAL\_DCMI\_RESET\_HANDLE\_STATE** **Description:**

- Reset DCMI handle state.

**Parameters:**

- \_\_HANDLE\_\_: specifies the DCMI handle.

**Return value:**

- None

**\_\_HAL\_DCMI\_ENABLE**

**Description:**

- Enable the DCMI.

**Parameters:**

- \_\_HANDLE\_\_: DCMI handle

**Return value:**

- None

**\_\_HAL\_DCMI\_DISABLE**

**Description:**

- Disable the DCMI.

**Parameters:**

- `__HANDLE__`: DCMI handle

**Return value:**

- None

`__HAL_DCMI_GET_FLAG`**Description:**

- Get the DCMI pending flags.

**Parameters:**

- `__HANDLE__`: DCMI handle
- `__FLAG__`: Get the specified flag. This parameter can be any combination of the following values:
  - `DCMI_FLAG_FRAMERI`: Frame capture complete flag mask
  - `DCMI_FLAG_OVFRI`: Overflow flag mask
  - `DCMI_FLAG_ERRRI`: Synchronization error flag mask
  - `DCMI_FLAG_VSYNCRI`: VSYNC flag mask
  - `DCMI_FLAG_LINERI`: Line flag mask

**Return value:**

- The state of FLAG.

`__HAL_DCMI_CLEAR_FLAG`**Description:**

- Clear the DCMI pending flags.

**Parameters:**

- `__HANDLE__`: DCMI handle
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DCMI_FLAG_FRAMERI`: Frame capture complete flag mask
  - `DCMI_FLAG_OVFRI`: Overflow flag mask
  - `DCMI_FLAG_ERRRI`: Synchronization error flag mask
  - `DCMI_FLAG_VSYNCRI`: VSYNC flag mask
  - `DCMI_FLAG_LINERI`: Line flag mask

**Return value:**

- None

`__HAL_DCMI_ENABLE_IT`**Description:**

- Enable the specified DCMI interrupts.

**Parameters:**

- `__HANDLE__`: DCMI handle
- `__INTERRUPT__`: specifies the DCMI interrupt sources to be enabled. This

parameter can be any combination of the following values:

- DCMI\_IT\_FRAME: Frame capture complete interrupt mask
- DCMI\_IT\_OVF: Overflow interrupt mask
- DCMI\_IT\_ERR: Synchronization error interrupt mask
- DCMI\_IT\_VSYNC: VSYNC interrupt mask
- DCMI\_IT\_LINE: Line interrupt mask

**Return value:**

- None

[\\_\\_HAL\\_DCMI\\_DISABLE\\_IT](#)

**Description:**

- Disable the specified DCMI interrupts.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): DCMI handle
- [\\_\\_INTERRUPT\\_\\_](#): specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI\_IT\_FRAME: Frame capture complete interrupt mask
  - DCMI\_IT\_OVF: Overflow interrupt mask
  - DCMI\_IT\_ERR: Synchronization error interrupt mask
  - DCMI\_IT\_VSYNC: VSYNC interrupt mask
  - DCMI\_IT\_LINE: Line interrupt mask

**Return value:**

- None

[\\_\\_HAL\\_DCMI\\_GET\\_IT\\_SOURCE](#)

**Description:**

- Check whether the specified DCMI interrupt has occurred or not.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): DCMI handle
- [\\_\\_INTERRUPT\\_\\_](#): specifies the DCMI interrupt source to check. This parameter can be one of the following values:
  - DCMI\_IT\_FRAME: Frame capture complete interrupt mask
  - DCMI\_IT\_OVF: Overflow interrupt mask
  - DCMI\_IT\_ERR: Synchronization error interrupt mask
  - DCMI\_IT\_VSYNC: VSYNC interrupt mask

- DCMI\_IT\_LINE: Line interrupt mask

**Return value:**

- The state of INTERRUPT.

**DCMI Extended Data Mode**

DCMI_EXTEND_DATA_8B	Interface captures 8-bit data on every pixel clock
DCMI_EXTEND_DATA_10B	Interface captures 10-bit data on every pixel clock
DCMI_EXTEND_DATA_12B	Interface captures 12-bit data on every pixel clock
DCMI_EXTEND_DATA_14B	Interface captures 14-bit data on every pixel clock

**DCMI Flags**

DCMI_FLAG_HSYNC
DCMI_FLAG_VSYNC
DCMI_FLAG_FNE
DCMI_FLAG_FRAMERI
DCMI_FLAG_OVFRI
DCMI_FLAG_ERRRI
DCMI_FLAG_VSYNCRI
DCMI_FLAG_LINERI
DCMI_FLAG_FRAMEMI
DCMI_FLAG_OVFM
DCMI_FLAG_ERRMI
DCMI_FLAG_VSYNCMI
DCMI_FLAG_LINEMI

**DCMI HSYNC Polarity**

DCMI_HSPOLARITY_LOW	Horizontal synchronization active Low
DCMI_HSPOLARITY_HIGH	Horizontal synchronization active High

**DCMI interrupt sources**

DCMI_IT_FRAME
DCMI_IT_OVF
DCMI_IT_ERR
DCMI_IT_VSYNC
DCMI_IT_LINE

**DCMI MODE JPEG**

DCMI_JPEG_DISABLE	Mode JPEG Disabled
DCMI_JPEG_ENABLE	Mode JPEG Enabled

**DCMI PIXCK Polarity**

DCMI_PCKPOLARITY_FALLING	Pixel clock active on Falling edge
--------------------------	------------------------------------

DCMI\_PCKPOLARITY\_RISING Pixel clock active on Rising edge

***DCMI Private Macros***

IS\_DCMI\_CAPTURE\_MODE

IS\_DCMI\_SYNCHRO

IS\_DCMI\_PCKPOLARITY

IS\_DCMI\_VSPOLARITY

IS\_DCMI\_HSPOLARITY

IS\_DCMI\_MODE\_JPEG

IS\_DCMI\_CAPTURE\_RATE

IS\_DCMI\_EXTENDED\_DATA

IS\_DCMI\_WINDOW\_COORDINATE

IS\_DCMI\_WINDOW\_HEIGHT

***DCMI Synchronization Mode***

DCMI\_SYNCHRO\_HARDWARE Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals

DCMI\_SYNCHRO\_EMBEDDED Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

***DCMI VSYNC Polarity***

DCMI\_VSPOLARITY\_LOW Vertical synchronization active Low

DCMI\_VSPOLARITY\_HIGH Vertical synchronization active High

***DCMI Window Coordinate***

DCMI\_WINDOW\_COORDINATE Window coordinate

***DCMI Window Height***

DCMI\_WINDOW\_HEIGHT Window Height

## 13 HAL DMA Generic Driver

### 13.1 DMA Firmware driver registers structures

#### 13.1.1 DMA\_InitTypeDef

##### 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\*\*](#)  
**Note:** The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Stream
- ***uint32\_t DMA\_InitTypeDef::Priority***  
Specifies the software priority for the DMAy Streamx. This parameter can be a value of [\*\*DMA\\_Priority\\_level\*\*](#)

- ***uint32\_t DMA\_InitTypeDef::FIFOMode***  
Specifies if the FIFO mode or Direct mode will be used for the specified stream. This parameter can be a value of [\*\*DMA\\_FIFO\\_direct\\_mode\*\*](#)  
**Note:**The Direct mode (FIFO mode disabled) cannot be used if the memory-to-memory data transfer is configured on the selected stream
- ***uint32\_t DMA\_InitTypeDef::FIFOThreshold***  
Specifies the FIFO threshold level. This parameter can be a value of [\*\*DMA\\_FIFO\\_threshold\\_level\*\*](#)
- ***uint32\_t DMA\_InitTypeDef::MemBurst***  
Specifies the Burst transfer configuration for the memory transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of [\*\*DMA\\_Memory\\_burst\*\*](#)  
**Note:**The burst mode is possible only if the address Increment mode is enabled.
- ***uint32\_t DMA\_InitTypeDef::PeriphBurst***  
Specifies the Burst transfer configuration for the peripheral transfers. It specifies the amount of data to be transferred in a single non interruptible transaction. This parameter can be a value of [\*\*DMA\\_Peripheral\\_burst\*\*](#)  
**Note:**The burst mode is possible only if the address Increment mode is enabled.

### 13.1.2 [\\_\\_DMA\\_HandleTypeDef](#)

#### 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***
- ***uint32\_t StreamBaseAddress***
- ***uint32\_t StreamIndex***

#### 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
- **`uint32_t __DMA_HandleTypeDef::StreamBaseAddress`**  
DMA Stream Base Address
- **`uint32_t __DMA_HandleTypeDef::StreamIndex`**  
DMA Stream Index

## 13.2 DMA Firmware driver API description

### 13.2.1 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

### 13.2.2 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.

This section contains the following APIs:

- `HAL_DMA_Init()`
- `HAL_DMA_DelInit()`

### 13.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request

This section contains the following APIs:

- `HAL_DMA_Start()`

- [\*HAL\\_DMA\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_DMA\\_Abort\(\)\*](#)
- [\*HAL\\_DMA\\_PollForTransfer\(\)\*](#)
- [\*HAL\\_DMA\\_IRQHandler\(\)\*](#)

### 13.2.4 State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code

This section contains the following APIs:

- [\*HAL\\_DMA\\_GetState\(\)\*](#)
- [\*HAL\\_DMA\\_GetError\(\)\*](#)

### 13.2.5 HAL\_DMA\_Init

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)</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>• HAL status</li> </ul>

### 13.2.6 HAL\_DMA\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)</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>• HAL status</li> </ul>

### 13.2.7 HAL\_DMA\_Start

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)</b>
Function Description	Starts 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> <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
- HAL status

### 13.2.8 HAL\_DMA\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)</b>
Function Description	Start the 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>DataLength:</b> The length of data to be transferred from source to destination</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 13.2.9 HAL\_DMA\_Abort

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * 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>• HAL status</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>

### 13.2.10 HAL\_DMA\_PollForTransfer

Function Name	<b>HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * 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>• HAL status</li> </ul>

### 13.2.11 HAL\_DMA\_IRQHandler

Function Name	<b>void HAL_DMA_IRQHandler (DMA_HandleTypeDef * 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>

### 13.2.12 HAL\_DMA\_GetState

Function Name	<b>HAL_DMA_StateTypeDef HAL_DMA_GetState (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>• HAL state</li> </ul>

### 13.2.13 HAL\_DMA\_GetError

Function Name	<b>uint32_t HAL_DMA_GetError (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>• DMA Error Code</li> </ul>

## 13.3 DMA Firmware driver defines

### 13.3.1 DMA

#### *DMA Channel selection*

DMA_CHANNEL_0	DMA Channel 0
DMA_CHANNEL_1	DMA Channel 1
DMA_CHANNEL_2	DMA Channel 2
DMA_CHANNEL_3	DMA Channel 3
DMA_CHANNEL_4	DMA Channel 4
DMA_CHANNEL_5	DMA Channel 5
DMA_CHANNEL_6	DMA Channel 6
DMA_CHANNEL_7	DMA Channel 7

#### *DMA Data transfer direction*

DMA_PERIPH_TO_MEMORY	Peripheral to memory direction
DMA_MEMORY_TO_PERIPH	Memory to peripheral direction
DMA_MEMORY_TO_MEMORY	Memory to memory direction

#### *DMA Error Code*

---

HAL_DMA_ERROR_NONE	No error
HAL_DMA_ERROR_TE	Transfer error
HAL_DMA_ERROR_FE	FIFO error
HAL_DMA_ERROR_DME	Direct Mode error
HAL_DMA_ERROR_TIMEOUT	Timeout error

**DMA FIFO direct mode**

DMA_FIFOMODE_DISABLE	FIFO mode disable
DMA_FIFOMODE_ENABLE	FIFO mode enable

**DMA FIFO threshold level**

DMA_FIFO_THRESHOLD_1QUARTERFULL	FIFO threshold 1 quart full configuration
DMA_FIFO_THRESHOLD_HALFFULL	FIFO threshold half full configuration
DMA_FIFO_THRESHOLD_3QUARTERSFULL	FIFO threshold 3 quarts full configuration
DMA_FIFO_THRESHOLD_FULL	FIFO threshold full configuration

**DMA flag definitions**

DMA_FLAG_FEIF0_4
DMA_FLAG_DMEIF0_4
DMA_FLAG_TEIF0_4
DMA_FLAG_HEIF0_4
DMA_FLAG_TCIF0_4
DMA_FLAG_FEIF1_5
DMA_FLAG_DMEIF1_5
DMA_FLAG_TEIF1_5
DMA_FLAG_HEIF1_5
DMA_FLAG_TCIF1_5
DMA_FLAG_FEIF2_6
DMA_FLAG_DMEIF2_6
DMA_FLAG_TEIF2_6
DMA_FLAG_HEIF2_6
DMA_FLAG_TCIF2_6
DMA_FLAG_FEIF3_7
DMA_FLAG_DMEIF3_7
DMA_FLAG_TEIF3_7
DMA_FLAG_HEIF3_7
DMA_FLAG_TCIF3_7

**DMA Handle index**

TIM_DMA_ID_UPDATE	Index of the DMA handle used for Update DMA requests
-------------------	--

---

TIM_DMA_ID_CC1	Index of the DMA handle used for Capture/Compare 1 DMA requests
TIM_DMA_ID_CC2	Index of the DMA handle used for Capture/Compare 2 DMA requests
TIM_DMA_ID_CC3	Index of the DMA handle used for Capture/Compare 3 DMA requests
TIM_DMA_ID_CC4	Index of the DMA handle used for Capture/Compare 4 DMA requests
TIM_DMA_ID_COMMUTATION	Index of the DMA handle used for Commutation DMA requests
TIM_DMA_ID_TRIGGER	Index of the DMA handle used for Trigger DMA requests

**DMA interrupt enable definitions**

DMA\_IT\_TC  
DMA\_IT\_HT  
DMA\_IT\_TE  
DMA\_IT\_DME  
DMA\_IT\_FE

**DMA Memory burst**

DMA\_MBURST\_SINGLE  
DMA\_MBURST\_INC4  
DMA\_MBURST\_INC8  
DMA\_MBURST\_INC16

**DMA Memory data size**

DMA\_MDATAALIGN\_BYTE      Memory data alignment: Byte  
DMA\_MDATAALIGN\_HALFWORD    Memory data alignment: HalfWord  
DMA\_MDATAALIGN\_WORD        Memory data alignment: Word

**DMA Memory incremented mode**

DMA\_MINC\_ENABLE    Memory increment mode enable  
DMA\_MINC\_DISABLE   Memory increment mode disable

**DMA mode**

DMA\_NORMAL        Normal mode  
DMA\_CIRCULAR      Circular mode  
DMA\_PFCTRL        Peripheral flow control mode

**DMA Peripheral burst**

DMA\_PBURST\_SINGLE  
DMA\_PBURST\_INC4  
DMA\_PBURST\_INC8  
DMA\_PBURST\_INC16

**DMA Peripheral data size**

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

**DMA Peripheral incremented mode**

DMA_PINC_ENABLE	Peripheral increment mode enable
DMA_PINC_DISABLE	Peripheral increment mode disable

**DMA Priority level**

DMA_PRIORITY_LOW	Priority level: Low
DMA_PRIORITY_MEDIUM	Priority level: Medium
DMA_PRIORITY_HIGH	Priority level: High
DMA_PRIORITY VERY HIGH	Priority level: Very High

**DMA Private Constants**

HAL\_TIMEOUT\_DMA\_ABORT

**DMA Private Macros**

IS_DMA_CHANNEL
IS_DMA_DIRECTION
IS_DMA_BUFFER_SIZE
IS_DMA_PERIPHERAL_INC_STATE
IS_DMA_MEMORY_INC_STATE
IS_DMA_PERIPHERAL_DATA_SIZE
IS_DMA_MEMORY_DATA_SIZE
IS_DMA_MODE
IS_DMA_PRIORITY
IS_DMA_FIFO_MODE_STATE
IS_DMA_FIFO_THRESHOLD
IS_DMA_MEMORY_BURST
IS_DMA_PERIPHERAL_BURST

## 14 HAL DMA Extension Driver

### 14.1 DMAEx Firmware driver API description

#### 14.1.1 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.

#### 14.1.2 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.

This section contains the following APIs:

- [\*\*HAL\\_DMAEx\\_MultiBufferStart\(\)\*\*](#)
- [\*\*HAL\\_DMAEx\\_MultiBufferStart\\_IT\(\)\*\*](#)
- [\*\*HAL\\_DMAEx\\_ChangeMemory\(\)\*\*](#)

#### 14.1.3 HAL\_DMAEx\_MultiBufferStart

Function Name	<code>HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t 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 source to destination</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 14.1.4 HAL\_DMAEx\_MultiBufferStart\_IT

Function Name	<code>HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t</code>
---------------	---

**DstAddress, uint32\_t SecondMemAddress, uint32\_t DataLength)**

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>• HAL status</li> </ul>

#### 14.1.5 HAL\_DMAEx\_ChangeMemory

Function Name	<b>HAL_StatusTypeDef HAL_DMAEx_ChangeMemory(DMA_HandleTypeDef * hdma, uint32_t Address, 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>

## 15 HAL ETH Generic Driver

### 15.1 ETH Firmware driver registers structures

#### 15.1.1 ETH\_InitTypeDef

##### 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\*](#)

#### 15.1.2 ETH\_MACInitTypeDef

##### 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\\_Unicast\\_Pause\\_Frame\\_Detect\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::ReceiveFlowControl***  
Enables or disables the MAC to decode the received Pause frame and disable its transmitter for a specified time (Pause Time) This parameter can be a value of [\*\*\*ETH\\_Receive\\_Flow\\_Control\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::TransmitFlowControl***  
Enables or disables the MAC to transmit Pause frames (Full-Duplex mode) or the MAC back-pressure operation (Half-Duplex mode) This parameter can be a value of [\*\*\*ETH\\_Transmit\\_Flow\\_Control\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::VLANTagComparison***  
Selects the 12-bit VLAN identifier or the complete 16-bit VLAN tag for comparison and filtering. This parameter can be a value of [\*\*\*ETH\\_VLAN\\_Tag\\_Comparison\*\*\*](#)
- ***uint32\_t ETH\_MACInitTypeDef::VLANTagIdentifier***  
Holds the VLAN tag identifier for receive frames

### 15.1.3 ETH\_DMADef

#### Data Fields

- ***uint32\_t DropTCPIPChecksumErrorFrame***
- ***uint32\_t ReceiveStoreForward***
- ***uint32\_t FlushReceivedFrame***
- ***uint32\_t TransmitStoreForward***
- ***uint32\_t TransmitThresholdControl***
- ***uint32\_t ForwardErrorFrames***
- ***uint32\_t ForwardUndersizedGoodFrames***
- ***uint32\_t ReceiveThresholdControl***
- ***uint32\_t SecondFrameOperate***
- ***uint32\_t AddressAlignedBeats***
- ***uint32\_t FixedBurst***
- ***uint32\_t RxDMABurstLength***
- ***uint32\_t TxDMABurstLength***
- ***uint32\_t EnhancedDescriptorFormat***
- ***uint32\_t DescriptorSkipLength***
- ***uint32\_t DMAArbitration***

#### Field Documentation

- ***uint32\_t ETH\_DMADef::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\_DMADef::ReceiveStoreForward***  
Enables or disables the Receive store and forward mode. This parameter can be a value of [\*\*\*ETH\\_Receive\\_Store\\_Forward\*\*\*](#)

- **`uint32_t ETH_DMADescTypeDef::Status`**  
Enables or disables the flushing of received frames. This parameter can be a value of [`ETH\_Flush\_Received\_Frame`](#)
- **`uint32_t ETH_DMADescTypeDef::ControlBufferSize`**  
Enables or disables Transmit store and forward mode. This parameter can be a value of [`ETH\_Transmit\_Store\_Forward`](#)
- **`uint32_t ETH_DMADescTypeDef::TransmitThresholdControl`**  
Selects or not the Transmit Threshold Control. This parameter can be a value of [`ETH\_Transmit\_Threshold\_Control`](#)
- **`uint32_t ETH_DMADescTypeDef::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_DMADescTypeDef::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_DMADescTypeDef::ReceiveThresholdControl`**  
Selects the threshold level of the Receive FIFO. This parameter can be a value of [`ETH\_Receive\_Threshold\_Control`](#)
- **`uint32_t ETH_DMADescTypeDef::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_DMADescTypeDef::AddressAlignedBeats`**  
Enables or disables the Address Aligned Beats. This parameter can be a value of [`ETH\_Address\_Aligned\_Beats`](#)
- **`uint32_t ETH_DMADescTypeDef::FixedBurst`**  
Enables or disables the AHB Master interface fixed burst transfers. This parameter can be a value of [`ETH\_Fixed\_Burst`](#)
- **`uint32_t ETH_DMADescTypeDef::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_DMADescTypeDef::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_DMADescTypeDef::EnhancedDescriptorFormat`**  
Enables the enhanced descriptor format. This parameter can be a value of [`ETH\_DMA\_Enhanced\_descriptor\_format`](#)
- **`uint32_t ETH_DMADescTypeDef::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_DMADescTypeDef::DMAArbitration`**  
Selects the DMA Tx/Rx arbitration. This parameter can be a value of [`ETH\_DMA\_Arbitration`](#)

#### 15.1.4 ETH\_DMADescTypeDef

##### Data Fields

- `_IO uint32_t Status`
- `uint32_t ControlBufferSize`
- `uint32_t Buffer1Addr`

- *uint32\_t Buffer2NextDescAddr*
- *uint32\_t ExtendedStatus*
- *uint32\_t Reserved1*
- *uint32\_t TimeStampLow*
- *uint32\_t TimeStampHigh*

#### 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

### 15.1.5 ETH\_DMARxFrameInfos

#### Data Fields

- *ETH\_DMADescTypeDef \* FSRxDesc*
- *ETH\_DMADescTypeDef \* LSRxDesc*
- *uint32\_t SegCount*
- *uint32\_t length*
- *uint32\_t buffer*

#### Field Documentation

- ***ETH\_DMADescTypeDef\* ETH\_DMARxFrameInfos::FSRxDesc***  
First Segment Rx Desc
- ***ETH\_DMADescTypeDef\* ETH\_DMARxFrameInfos::LSRxDesc***  
Last Segment Rx Desc
- ***uint32\_t ETH\_DMARxFrameInfos::SegCount***  
Segment count
- ***uint32\_t ETH\_DMARxFrameInfos::length***  
Frame length
- ***uint32\_t ETH\_DMARxFrameInfos::buffer***  
Frame buffer

## 15.1.6 ETH\_HandleTypeDef

### Data Fields

- *ETH\_TypeDef \* Instance*
- *ETH\_InitTypeDef Init*
- *uint32\_t LinkStatus*
- *ETH\_DMADescTypeDef \* RxDesc*
- *ETH\_DMADescTypeDef \* TxDesc*
- *ETH\_DMARxFrameInfos RxFrameInfos*
- *\_IO HAL\_ETH\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*

### Field Documentation

- ***ETH\_TypeDef\* ETH\_HandleTypeDef::Instance***  
Register base address
- ***ETH\_InitTypeDef ETH\_HandleTypeDef::Init***  
Ethernet Init Configuration
- ***uint32\_t ETH\_HandleTypeDef::LinkStatus***  
Ethernet link status
- ***ETH\_DMADescTypeDef\* ETH\_HandleTypeDef::RxDesc***  
Rx descriptor to Get
- ***ETH\_DMADescTypeDef\* ETH\_HandleTypeDef::TxDesc***  
Tx descriptor to Set
- ***ETH\_DMARxFrameInfos ETH\_HandleTypeDef::RxFrameInfos***  
last Rx frame infos
- ***\_IO HAL\_ETH\_StateTypeDef ETH\_HandleTypeDef::State***  
ETH communication state
- ***HAL\_LockTypeDef ETH\_HandleTypeDef::Lock***  
ETH Lock

## 15.2 ETH Firmware driver API description

### 15.2.1 How to use this driver

1. Declare a ETH\_HandleTypeDef handle structure, for example: ETH\_HandleTypeDef heth;
2. Fill parameters of Init structure in heth handle
3. Call HAL\_ETH\_Init() API to initialize the Ethernet peripheral (MAC, DMA, ...)
4. Initialize the ETH low level resources through the HAL\_ETH\_MspInit() API:
  - a. Enable the Ethernet interface clock using
    - `__HAL_RCC_ETHMAC_CLK_ENABLE();`
    - `__HAL_RCC_ETHMACTX_CLK_ENABLE();`
    - `__HAL_RCC_ETHMACRX_CLK_ENABLE();`
  - b. Initialize the related GPIO clocks
  - c. Configure Ethernet pin-out
  - d. Configure Ethernet NVIC interrupt (IT mode)
5. Initialize Ethernet DMA Descriptors in chain mode and point to allocated buffers:
  - a. `HAL_ETH_DMATxDescListInit();` for Transmission process

- b. HAL\_ETH\_DMARxDescListInit(); for Reception process
6. Enable MAC and DMA transmission and reception:
  - a. HAL\_ETH\_Start();
7. Prepare ETH DMA TX Descriptors and give the hand to ETH DMA to transfer the frame to MAC TX FIFO:
  - a. HAL\_ETH\_TransmitFrame();
8. Poll for a received frame in ETH RX DMA Descriptors and get received frame parameters
  - a. HAL\_ETH\_GetReceivedFrame(); (should be called into an infinite loop)
9. Get a received frame when an ETH RX interrupt occurs:
  - a. HAL\_ETH\_GetReceivedFrame\_IT(); (called in IT mode only)
10. Communicate with external PHY device:
  - a. Read a specific register from the PHY HAL\_ETH\_ReadPHYRegister();
  - b. Write data to a specific RHY register: HAL\_ETH\_WritePHYRegister();
11. Configure the Ethernet MAC after ETH peripheral initialization  
HAL\_ETH\_ConfigMAC(); all MAC parameters should be filled.
12. Configure the Ethernet DMA after ETH peripheral initialization  
HAL\_ETH\_ConfigDMA(); all DMA parameters should be filled.

### 15.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the Ethernet peripheral
- De-initialize the Ethernet peripheral

This section contains the following APIs:

- [\*\*HAL\\_ETH\\_Init\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_Delinit\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_DMATxDescListInit\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_DMARxDescListInit\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_MspDelinit\(\)\*\*](#)

### 15.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a frame HAL\_ETH\_TransmitFrame();
- Receive a frame HAL\_ETH\_GetReceivedFrame();  
HAL\_ETH\_GetReceivedFrame\_IT();
- Read from an External PHY register HAL\_ETH\_ReadPHYRegister();
- Write to an External PHY register HAL\_ETH\_WritePHYRegister();

This section contains the following APIs:

- [\*\*HAL\\_ETH\\_TransmitFrame\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_GetReceivedFrame\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_GetReceivedFrame\\_IT\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_IRQHandler\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_TxCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_RxCpltCallback\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_ErrorCallback\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_ReadPHYRegister\(\)\*\*](#)
- [\*\*HAL\\_ETH\\_WritePHYRegister\(\)\*\*](#)

### 15.2.4 Peripheral Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

- `HAL\_ETH\_Start\(\)`
- `HAL\_ETH\_Stop\(\)`
- `HAL\_ETH\_ConfigMAC\(\)`
- `HAL\_ETH\_ConfigDMA\(\)`

### 15.2.5 Peripheral State functions

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

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

This section contains the following APIs:

- `HAL\_ETH\_GetState\(\)`

### 15.2.6 HAL\_ETH\_Init

Function Name	<code>HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)</code>
Function Description	Initializes the Ethernet MAC and DMA according to default parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth:</b> pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.7 HAL\_ETH\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_ETH_DelInit (ETH_HandleTypeDef * heth)</code>
Function Description	De-Initializes the ETH peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth:</b> pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.8 HAL\_ETH\_DMATxDescListInit

Function Name	<code>HAL_StatusTypeDef HAL_ETH_DMATxDescListInit (ETH_HandleTypeDef * heth, ETH_DMADescTypeDef * DMATxDescTab, uint8_t * TxBuff, uint32_t TxBuffCount)</code>
Function Description	Initializes the DMA Tx descriptors in chain mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth:</b> pointer to a <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> <li>• <b>DMATxDescTab:</b> Pointer to the first Tx desc list</li> </ul>

- **TxBuff:** Pointer to the first TxBuffer list
- **TxBuffCount:** Number of the used Tx desc in the list
- HAL status

Return values

**15.2.9 HAL\_ETH\_DMARxDescListInit**

Function Name

**HAL\_StatusTypeDef HAL\_ETH\_DMARxDescListInit  
(ETH\_HandleTypeDef \* heth, ETH\_DMADescTypeDef \*  
DMARxDescTab, uint8\_t \* RxBuff, uint32\_t RxBuffCount)**

Function Description

Initializes the DMA Rx descriptors in chain mode.

Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **DMARxDescTab:** Pointer to the first Rx desc list
- **RxBuff:** Pointer to the first RxBuffer list
- **RxBuffCount:** Number of the used Rx desc in the list

Return values

- HAL status

**15.2.10 HAL\_ETH\_MspInit**

Function Name

**void HAL\_ETH\_MspInit (ETH\_HandleTypeDef \* heth)**

Function Description

Initializes the ETH MSP.

Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- None

**15.2.11 HAL\_ETH\_MspDeInit**

Function Name

**void HAL\_ETH\_MspDeInit (ETH\_HandleTypeDef \* heth)**

Function Description

Deinitializes ETH MSP.

Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module

Return values

- None

**15.2.12 HAL\_ETH\_TransmitFrame**

Function Name

**HAL\_StatusTypeDef HAL\_ETH\_TransmitFrame  
(ETH\_HandleTypeDef \* heth, uint32\_t FrameLength)**

Function Description

Sends an Ethernet frame.

Parameters

- **heth:** pointer to a ETH\_HandleTypeDef structure that contains the configuration information for ETHERNET module
- **FrameLength:** Amount of data to be sent

Return values

- HAL status

**15.2.13 HAL\_ETH\_GetReceivedFrame**

Function Name

**HAL\_StatusTypeDef HAL\_ETH\_GetReceivedFrame  
(ETH\_HandleTypeDef \* heth)**

	Function Description	Checks for received frames.
	Parameters	<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>• HAL status</li> </ul>
<b>15.2.14 HAL_ETH_GetReceivedFrame_IT</b>		
	Function Name	<b>HAL_StatusTypeDef HAL_ETH_GetReceivedFrame_IT (ETH_HandleTypeDef * 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>• HAL status</li> </ul>
<b>15.2.15 HAL_ETH_IRQHandler</b>		
	Function Name	<b>void HAL_ETH_IRQHandler (ETH_HandleTypeDef * heth)</b>
	Function Description	This function handles ETH interrupt request.
	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>• HAL status</li> </ul>
<b>15.2.16 HAL_ETH_TxCpltCallback</b>		
	Function Name	<b>void HAL_ETH_TxCpltCallback (ETH_HandleTypeDef * heth)</b>
	Function Description	Tx Transfer completed callbacks.
	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>
<b>15.2.17 HAL_ETH_RxCpltCallback</b>		
	Function Name	<b>void HAL_ETH_RxCpltCallback (ETH_HandleTypeDef * heth)</b>
	Function Description	Rx Transfer completed callbacks.
	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>
<b>15.2.18 HAL_ETH_ErrorCallback</b>		
	Function Name	<b>void HAL_ETH_ErrorCallback (ETH_HandleTypeDef * heth)</b>
	Function Description	Ethernet transfer error callbacks.
	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>

### 15.2.19 HAL\_ETH\_ReadPHYRegister

Function Name	<code>HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t * RegValue)</code>
Function Description	Reads 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 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>• HAL status</li> </ul>

### 15.2.20 HAL\_ETH\_WritePHYRegister

Function Name	<code>HAL_StatusTypeDef HAL_ETH_WritePHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t RegValue)</code>
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 PHY register could be written depending on the used PHY</li> <li>• <b>RegValue:</b> the value to write</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.21 HAL\_ETH\_Start

Function Name	<code>HAL_StatusTypeDef HAL_ETH_Start (ETH_HandleTypeDef * heth)</code>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.22 HAL\_ETH\_Stop

Function Name	<code>HAL_StatusTypeDef HAL_ETH_Stop (ETH_HandleTypeDef * heth)</code>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.23 HAL\_ETH\_ConfigMAC

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ConfigMAC (ETH_HandleTypeDef * heth, ETH_MACInitTypeDef * macconf)</b>
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>• HAL status</li> </ul>

### 15.2.24 HAL\_ETH\_ConfigDMA

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ConfigDMA (ETH_HandleTypeDef * heth, ETH_DMAMailInitTypeDef * dmaconf)</b>
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>• HAL status</li> </ul>

### 15.2.25 HAL\_ETH\_GetState

Function Name	<b>HAL_ETH_StateTypeDef HAL_ETH_GetState (ETH_HandleTypeDef * heth)</b>
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 contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

## 15.3 ETH Firmware driver defines

### 15.3.1 ETH

#### *ETH Address Aligned Beats*

ETH\_ADDRESSALIGNEDBEATS\_ENABLE

ETH\_ADDRESSALIGNEDBEATS\_DISABLE

#### *ETH Automatic Pad CRC Strip*

ETH\_AUTOMATICPADCRCSTRIP\_ENABLE

ETH\_AUTOMATICPADCRCSTRIP\_DISABLE

#### *ETH AutoNegotiation*

ETH\_AUTONEGOTIATION\_ENABLE

ETH\_AUTONEGOTIATION\_DISABLE

#### *ETH Back Off Limit*

ETH\_BACKOFFLIMIT\_10



ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_2\_1  
 ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_3\_1  
 ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_4\_1  
 ETH\_DMAARBITRATION\_RXPRIORTX

***ETH DMA Enhanced descriptor format***

ETH\_DMAENHANCEDDESCRIPTOR\_ENABLE  
 ETH\_DMAENHANCEDDESCRIPTOR\_DISABLE

***ETH DMA Flags***

ETH_DMA_FLAG_TST	Time-stamp trigger interrupt (on DMA)
ETH_DMA_FLAG_PMT	PMT interrupt (on DMA)
ETH_DMA_FLAG_MMC	MMC interrupt (on DMA)
ETH_DMA_FLAG_DATATRANSFERERROR	Error bits 0-Rx DMA, 1-Tx DMA
ETH_DMA_FLAG_READWRITEERROR	Error bits 0-write transfer, 1-read transfer
ETH_DMA_FLAG_ACCESSERROR	Error bits 0-data buffer, 1-desc. access
ETH_DMA_FLAG_NIS	Normal interrupt summary flag
ETH_DMA_FLAG_AIS	Abnormal interrupt summary flag
ETH_DMA_FLAG_ER	Early receive flag
ETH_DMA_FLAG_FBE	Fatal bus error flag
ETH_DMA_FLAG_ET	Early transmit flag
ETH_DMA_FLAG_RWT	Receive watchdog timeout flag
ETH_DMA_FLAG_RPS	Receive process stopped flag
ETH_DMA_FLAG_RBU	Receive buffer unavailable flag
ETH_DMA_FLAG_R	Receive flag
ETH_DMA_FLAG_TU	Underflow flag
ETH_DMA_FLAG_RO	Overflow flag
ETH_DMA_FLAG_TJT	Transmit jabber timeout flag
ETH_DMA_FLAG_TBU	Transmit buffer unavailable flag
ETH_DMA_FLAG_TPS	Transmit process stopped flag
ETH_DMA_FLAG_T	Transmit flag

***ETH DMA Interrupts***

ETH_DMA_IT_TST	Time-stamp trigger interrupt (on DMA)
ETH_DMA_IT_PMT	PMT interrupt (on DMA)
ETH_DMA_IT_MMC	MMC interrupt (on DMA)
ETH_DMA_IT_NIS	Normal interrupt summary
ETH_DMA_IT_AIS	Abnormal interrupt summary
ETH_DMA_IT_ER	Early receive interrupt

ETH_DMA_IT_FBE	Fatal bus error interrupt
ETH_DMA_IT_ET	Early transmit interrupt
ETH_DMA_IT_RWT	Receive watchdog timeout interrupt
ETH_DMA_IT_RPS	Receive process stopped interrupt
ETH_DMA_IT_RBU	Receive buffer unavailable interrupt
ETH_DMA_IT_R	Receive interrupt
ETH_DMA_IT_TU	Underflow interrupt
ETH_DMA_IT_RO	Overflow interrupt
ETH_DMA_IT_TJT	Transmit jabber timeout interrupt
ETH_DMA_IT_TBU	Transmit buffer unavailable interrupt
ETH_DMA_IT_TPS	Transmit process stopped interrupt
ETH_DMA_IT_T	Transmit interrupt

***ETH DMA overflow***

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

***ETH DMA receive process state***

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

***ETH DMA RX Descriptor***

ETH_DMARXDESC_OWN	OWN bit: descriptor is owned by DMA engine
ETH_DMARXDESC_AFM	DA Filter Fail for the rx frame
ETH_DMARXDESC_FL	Receive descriptor frame length
ETH_DMARXDESC_ES	Error summary: OR of the following bits: DE    OE    IPC    LC    RWT    RE

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

	d Checksum Error: Rx MAC address matched/ Rx Payload Checksum Error
ETH_DMARXDESCDIC	Disable Interrupt on Completion
ETH_DMARXDESCRBS2	Receive Buffer2 Size
ETH_DMARXDESCRER	Receive End of Ring
ETH_DMARXDESCRCH	Second Address Chained
ETH_DMARXDESCRBS1	Receive Buffer1 Size
ETH_DMARXDESCB1AP	Buffer1 Address Pointer
ETH_DMARXDESCB2AP	Buffer2 Address Pointer
ETH_DMAPTPRXDESCPTPV	
ETH_DMAPTPRXDESCPTPFT	
ETH_DMAPTPRXDESCPTPMT	
ETH_DMAPTPRXDESCPTPMT_SYNC	
ETH_DMAPTPRXDESCPTPMT_FOLLOWUP	
ETH_DMAPTPRXDESCPTPMT_DELAYREQ	
ETH_DMAPTPRXDESCPTPMT_DELAYRESP	
ETH_DMAPTPRXDESCPTPMT_PDELAYREQ_ANNOUNCE	
ETH_DMAPTPRXDESCPTPMT_PDELAYRESP_MANAG	
ETH_DMAPTPRXDESCPTPMT_PDELAYRESPFOLLOWUP_SIGNAL	
ETH_DMAPTPRXDESCIPV6PR	
ETH_DMAPTPRXDESCIPV4PR	
ETH_DMAPTPRXDESCIPCB	
ETH_DMAPTPRXDESCIPPE	
ETH_DMAPTPRXDESCIPHE	
ETH_DMAPTPRXDESCIPPT	
ETH_DMAPTPRXDESCIPPT_UDP	
ETH_DMAPTPRXDESCIPPT_TCP	
ETH_DMAPTPRXDESCIPPT_ICMP	
ETH_DMAPTPRXDESCRTSL	

**ETH\_DMADMAPTRXDESC\_RTSH**

***ETH DMA Rx descriptor buffers***

**ETH\_DMARXDESC\_BUFFER1** DMA Rx Desc Buffer1

**ETH\_DMARXDESC\_BUFFER2** DMA Rx Desc Buffer2

***ETH DMA transmit process state***

<b>ETH_DMA_TRANSMITPROCESS_STOPPED</b>	Stopped - Reset or Stop Tx Command issued
--	---

<b>ETH_DMA_TRANSMITPROCESS_FETCHING</b>	Running - fetching the Tx descriptor
---	--------------------------------------

<b>ETH_DMA_TRANSMITPROCESS_WAITING</b>	Running - waiting for status
--	------------------------------

<b>ETH_DMA_TRANSMITPROCESS_READING</b>	Running - reading the data from host memory
--	---

<b>ETH_DMA_TRANSMITPROCESS_SUSPENDED</b>	Suspended - Tx Descriptor unavailable
--	---------------------------------------

<b>ETH_DMA_TRANSMITPROCESS_CLOSING</b>	Running - closing Rx descriptor
--	---------------------------------

***ETH DMA TX Descriptor***

<b>ETH_DMATXDESC_OWN</b>	OWN bit: descriptor is owned by DMA engine
--------------------------	--

<b>ETH_DMATXDESC_IC</b>	Interrupt on Completion
-------------------------	-------------------------

<b>ETH_DMATXDESC_LS</b>	Last Segment
-------------------------	--------------

<b>ETH_DMATXDESC_FS</b>	First Segment
-------------------------	---------------

<b>ETH_DMATXDESC_DC</b>	Disable CRC
-------------------------	-------------

<b>ETH_DMATXDESC_DP</b>	Disable Padding
-------------------------	-----------------

<b>ETH_DMATXDESC_TTSE</b>	Transmit Time Stamp Enable
---------------------------	----------------------------

<b>ETH_DMATXDESC_CIC</b>	Checksum Insertion Control: 4 cases
--------------------------	-------------------------------------

<b>ETH_DMATXDESC_CIC_BYPASS</b>	Do Nothing: Checksum Engine is bypassed
---------------------------------	---

<b>ETH_DMATXDESC_CIC_IPV4HEADER</b>	IPv4 header Checksum Insertion
-------------------------------------	--------------------------------

<b>ETH_DMATXDESC_CIC_TCPUDPICMP_SEGMENT</b>	TCP/UDP/ICMP Checksum Insertion calculated over segment only
---	--

<b>ETH_DMATXDESC_CIC_TCPUDPICMP_FULL</b>	TCP/UDP/ICMP Checksum Insertion fully calculated
--	--

<b>ETH_DMATXDESC_TER</b>	Transmit End of Ring
--------------------------	----------------------

<b>ETH_DMATXDESC_TCH</b>	Second Address Chained
--------------------------	------------------------

<b>ETH_DMATXDESC_TTSS</b>	Tx Time Stamp Status
---------------------------	----------------------

<b>ETH_DMATXDESC_IHE</b>	IP Header Error
--------------------------	-----------------

<b>ETH_DMATXDESC_ES</b>	Error summary: OR of the following bits: UE    ED    EC    LCO    NC    LCA    FF    JT
-------------------------	---

ETH_DMATXDESC_JT	Jabber Timeout
ETH_DMATXDESC_FF	Frame Flushed: DMA/MTL flushed the frame due to SW flush
ETH_DMATXDESC_PCE	Payload Checksum Error
ETH_DMATXDESC_LCA	Loss of Carrier: carrier lost during transmission
ETH_DMATXDESC_NC	No Carrier: no carrier signal from the transceiver
ETH_DMATXDESC_LCO	Late Collision: transmission aborted due to collision
ETH_DMATXDESC_EC	Excessive Collision: transmission aborted after 16 collisions
ETH_DMATXDESC_VF	VLAN Frame
ETH_DMATXDESC_CC	Collision Count
ETH_DMATXDESC_ED	Excessive Deferral
ETH_DMATXDESC_UF	Underflow Error: late data arrival from the memory
ETH_DMATXDESC_DB	Deferred Bit
ETH_DMATXDESC_TBS2	Transmit Buffer2 Size
ETH_DMATXDESC_TBS1	Transmit Buffer1 Size
ETH_DMATXDESC_B1AP	Buffer1 Address Pointer
ETH_DMATXDESC_B2AP	Buffer2 Address Pointer
ETH_DMAPPTXDESC_TTSL	
ETH_DMAPPTXDESC_TTSH	

***ETH DMA Tx descriptor Checksum Insertion Control***

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

***ETH DMA Tx descriptor segment***

ETH_DMATXDESC_LASTSEGMENTS	Last Segment
ETH_DMATXDESC_FIRSTSEGMENT	First Segment

***ETH Drop TCP IP Checksum Error Frame***

ETH\_DROPTCPIPCHECKSUMERRORFRAME\_ENABLE

ETH\_DROPTCIPCHECKSUMERRORFRAME\_DISABLE

**ETH Duplex Mode**

ETH\_MODE\_FULLDUPLEX

ETH\_MODE\_HALFDUPLEX

**ETH Exported Macros**

`__HAL_ETH_RESET_HANDLE_STATE`

**Description:**

- Reset ETH handle state.

**Parameters:**

- `__HANDLE__`: specifies the ETH handle.

**Return value:**

- None

`__HAL_ETH_DMATXDESC_GET_FLAG`

**Description:**

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

**Parameters:**

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

**Return value:**

- the: `ETH_DMATxDescFlag` (SET or RESET).

`__HAL_ETH_DMARXDESC_GET_FLAG`

**Description:**

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

**Parameters:**

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

**Return value:**

- the: `ETH_DMATxDescFlag` (SET or RESET).

`__HAL_ETH_DMARXDESC_ENABLE_IT`

**Description:**

- Enables the specified DMA Rx Desc receive interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

\_\_HAL\_ETH\_DMARXDESC\_DISABLE\_IT

**Description:**

- Disables the specified DMA Rx Desc receive interrupt.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle

**Return value:**

- None

\_\_HAL\_ETH\_DMARXDESC\_SET\_OWN\_BIT

**Description:**

- Set the specified DMA Rx Desc Own bit.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle

**Return value:**

- None

\_\_HAL\_ETH\_DMATXDESC\_GET\_COLLISION\_COUNT

**Description:**

- Returns the specified ETHERNET DMA Tx Desc collision count.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle

**Return value:**

- The: Transmit descriptor collision counter value.

\_\_HAL\_ETH\_DMATXDESC\_SET\_OWN\_BIT

**Description:**

- Set the specified DMA Tx Desc Own bit.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle

**Return value:**

- None

\_\_HAL\_ETH\_DMATXDESC\_ENABLE\_IT

**Description:**

- Enables the specified DMA Tx Desc Transmit interrupt.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle

**Return value:**

- None

\_\_HAL\_ETH\_DMATXDESC\_DISABLE\_IT

**Description:**

- Disables the specified DMA Tx Desc

Transmit interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`_HAL_ETH_DMATXDESC_CHECKSUM  
_INSERTION`

**Description:**

- Selects the specified ETHERNET DMA Tx Desc Checksum Insertion.

**Parameters:**

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

**Return value:**

- None

`_HAL_ETH_DMATXDESC_CRC_ENABLE`

**Description:**

- Enables the DMA Tx Desc CRC.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`_HAL_ETH_DMATXDESC_CRC_DISABLE`

**Description:**

- Disables the DMA Tx Desc CRC.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

<code>__HAL_ETH_DMATXDESC_SHORT_FRA ME_PADDING_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Enables the DMA Tx Desc padding for frame shorter than 64 bytes.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_ETH_DMATXDESC_SHORT_FRA ME_PADDING_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Disables the DMA Tx Desc padding for frame shorter than 64 bytes.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_ETH_MAC_ENABLE_IT</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Enables the specified ETHERNET MAC interrupts.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle</li><li><code>__INTERRUPT__</code>: specifies the ETHERNET MAC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:<ul style="list-style-type: none"><li>- <code>ETH_MAC_IT_TST</code> : Time stamp trigger interrupt</li><li>- <code>ETH_MAC_IT_PMT</code> : PMT interrupt</li></ul></li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_ETH_MAC_DISABLE_IT</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Disables the specified ETHERNET MAC interrupts.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle</li><li><code>__INTERRUPT__</code>: specifies the ETHERNET MAC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:<ul style="list-style-type: none"><li>- <code>ETH_MAC_IT_TST</code> : Time stamp trigger interrupt</li><li>- <code>ETH_MAC_IT_PMT</code> : PMT interrupt</li></ul></li></ul> <b>Return value:</b>

- None

`_HAL_ETH_INITIATE_PAUSE_CONTROL_FRAME`

**Description:**

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

**Parameters:**

- `_HANDLE_`: ETH Handle

**Return value:**

- None

`_HAL_ETH_GET_FLOW_CONTROL_BUSY_STATUS`

**Description:**

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

**Parameters:**

- `_HANDLE_`: ETH Handle

**Return value:**

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

`_HAL_ETH_BACK_PRESSURE_ACTIVATION_ENABLE`

**Description:**

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

**Parameters:**

- `_HANDLE_`: ETH Handle

**Return value:**

- None

`_HAL_ETH_BACK_PRESSURE_ACTIVATION_DISABLE`

**Description:**

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

**Parameters:**

- `_HANDLE_`: ETH Handle

**Return value:**

- None

`_HAL_ETH_MAC_GET_FLAG`

**Description:**

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

**Parameters:**

- `_HANDLE_`: ETH Handle
- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - `ETH_MAC_FLAG_TST` : Time stamp trigger flag

- ETH\_MAC\_FLAG\_MMCT : MMC transmit flag
- ETH\_MAC\_FLAG\_MMCR : MMC receive flag
- ETH\_MAC\_FLAG\_MMIC : MMC flag
- ETH\_MAC\_FLAG\_PMT : PMT flag

**Return value:**

- The state of ETHERNET MAC flag.

[\\_\\_HAL\\_ETH\\_DMA\\_ENABLE\\_IT](#)

**Description:**

- Enables the specified ETHERNET DMA interrupts.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): ETH Handle
- [\\_\\_INTERRUPT\\_\\_](#): specifies the ETHERNET DMA interrupt sources to be enabled

**Return value:**

- None

[\\_\\_HAL\\_ETH\\_DMA\\_DISABLE\\_IT](#)

**Description:**

- Disables the specified ETHERNET DMA interrupts.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): ETH Handle
- [\\_\\_INTERRUPT\\_\\_](#): specifies the ETHERNET DMA interrupt sources to be disabled.

**Return value:**

- None

[\\_\\_HAL\\_ETH\\_DMA\\_CLEAR\\_IT](#)

**Description:**

- Clears the ETHERNET DMA IT pending bit.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): ETH Handle
- [\\_\\_INTERRUPT\\_\\_](#): specifies the interrupt pending bit to clear.

**Return value:**

- None

[\\_\\_HAL\\_ETH\\_DMA\\_GET\\_FLAG](#)

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_FLAG\_\_: specifies the flag to check.

**Return value:**

- The: new state of ETH\_DMA\_FLAG (SET or RESET).

[\\_\\_HAL\\_ETH\\_DMA\\_CLEAR\\_FLAG](#)**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_FLAG\_\_: specifies the flag to clear.

**Return value:**

- The: new state of ETH\_DMA\_FLAG (SET or RESET).

[\\_\\_HAL\\_ETH\\_GET\\_DMA\\_OVERFLOW\\_STATUS](#)**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_OVERFLOW\_\_: specifies the DMA overflow flag to check. This parameter can be one of the following values:
  - ETH\_DMA\_OVERFLOW\_RXFIFO\_COUNTER : Overflow for FIFO Overflows Counter
  - ETH\_DMA\_OVERFLOW\_MISSED\_FRAMECOUNTER : Overflow for Buffer Unavailable Missed Frame Counter

**Return value:**

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

[\\_\\_HAL\\_ETH\\_SET\\_RECEIVE\\_WATCHDOG\\_TIMER](#)**Description:**

- Set the DMA Receive status watchdog timer register value.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle
- \_\_VALUE\_\_: DMA Receive status watchdog timer register value

**Return value:**

- None

[\\_\\_HAL\\_ETH\\_GLOBAL\\_UNICAST\\_WAKE](#)**Description:**

<code>UP_ENABLE</code>	<ul style="list-style-type: none"><li>Enables any unicast packet filtered by the MAC address recognition to be a wake-up frame.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_ETH_GLOBAL_UNICAST_WAKEUP_ENABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Disables any unicast packet filtered by the MAC address recognition to be a wake-up frame.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_ETH_WAKEUP_FRAME_DETECT_ENABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Enables the MAC Wake-Up Frame Detection.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_ETH_WAKEUP_FRAME_DETECT_DISABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Disables the MAC Wake-Up Frame Detection.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_ETH_MAGIC_PACKET_DETECT_ENABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Enables the MAC Magic Packet Detection.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: ETH Handle.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_ETH_MAGIC_PACKET_DETECT_DISABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Disables the MAC Magic Packet</li></ul>

Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_POWER_DOWN_ENABLE`

- Enables the MAC Power Down.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_POWER_DOWN_DISABLE`

- Disables the MAC Power Down.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_GET_PMT_FLAG_STATUS`

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

**Parameters:**

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

**Return value:**

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

`__HAL_ETH_MM_COUNTER_FULL_RESET`

**Description:**

- Preset and Initialize the MMC counters to almost-full value: 0xFFFF\_FFF0 (full - 16)

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_MMC_COUNTER_HALF_PR  
ESET`

**Description:**

- Preset and Initialize the MMC counters to almost-half value: 0x7FFF\_FFF0 (half - 16)

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_MMC_COUNTER_FREEZE_  
ENABLE`

**Description:**

- Enables the MMC Counter Freeze.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_MMC_COUNTER_FREEZE_  
DISABLE`

**Description:**

- Disables the MMC Counter Freeze.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_ETH_MMC_RESET_ONREA  
D_ENABLE`

**Description:**

- Enables the MMC Reset On Read.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_ETH_MMC_RESET_ONREA  
D_DISABLE`

**Description:**

- Disables the MMC Reset On Read.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_ETH_MMC_COUNTER_RO  
LLOVER_ENABLE`

**Description:**

- Enables the MMC Counter Stop

Rollover.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**Description:**

- Disables the MMC Counter Stop Rollover.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**Description:**

- Resets the MMC Counters.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**Description:**

- Enables the specified ETHERNET MMC Rx interrupts.

**Parameters:**

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

**Return value:**

- None

**Description:**

- Disables the specified ETHERNET MMC Rx interrupts.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle.
- \_\_INTERRUPT\_\_: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH\_MMC\_IT\_RGUF : When Rx good unicast frames counter reaches half the maximum value
  - ETH\_MMC\_IT\_RFAE : When Rx alignment error counter reaches half the maximum value
  - ETH\_MMC\_IT\_RFCE : When Rx crc error counter reaches half the maximum value

**Return value:**

- None

[\\_\\_HAL\\_ETH\\_MMC\\_TX\\_IT\\_ENABLE](#)**Description:**

- Enables the specified ETHERNET MMC Tx interrupts.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle.
- \_\_INTERRUPT\_\_: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH\_MMC\_IT\_TGF : When Tx good frame counter reaches half the maximum value
  - ETH\_MMC\_IT\_TGFMSC: When Tx good multi col counter reaches half the maximum value
  - ETH\_MMC\_IT\_TGFSC : When Tx good single col counter reaches half the maximum value

**Return value:**

- None

[\\_\\_HAL\\_ETH\\_MMC\\_TX\\_IT\\_DISABLE](#)**Description:**

- Disables the specified ETHERNET MMC Tx interrupts.

**Parameters:**

- \_\_HANDLE\_\_: ETH Handle.
- \_\_INTERRUPT\_\_: specifies the ETHERNET MMC interrupt sources to be enabled or disabled. This parameter can be one of the following values:
  - ETH\_MMC\_IT\_TGF : When Tx good frame counter reaches half

- the maximum value
- ETH\_MMC\_IT\_TGFMSC: When Tx good multi col counter reaches half the maximum value
- ETH\_MMC\_IT\_TGFSC : When Tx good single col counter reaches half the maximum value

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_IT`

**Description:**

- Enables the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_IT`

**Description:**

- Disables the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_EVENT`

**Description:**

- Enable event on ETH External event line.

**Return value:**

- None.

`__HAL_ETH_WAKEUP_EXTI_DISABLE_EVENT`

**Description:**

- Disable event on ETH External event line.

**Return value:**

- None.

`__HAL_ETH_WAKEUP_EXTI_GET_FLAG`

**Description:**

- Get flag of the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_CLEAR_FLAG`

**Description:**

- Clear flag of the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_EDGE_TRIGGER`

**Description:**

- Enables rising edge trigger to the ETH

External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_  
RISING_EDGE_TRIGGER`

**Description:**

- Disables the rising edge trigger to the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_  
FALLING_EDGE_TRIGGER`

**Description:**

- Enables falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_  
FALLING_EDGE_TRIGGER`

**Description:**

- Disables falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_  
FALLINGRISING_TRIGGER`

**Description:**

- Enables rising/falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_  
FALLINGRISING_TRIGGER`

**Description:**

- Disables rising/falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_GENERAT  
E_SWIT`

**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.

### ***ETH EXTI LINE WAKEUP***

`ETH_EXTI_LINE_WAKEUP` External interrupt line 19 Connected to the ETH EXTI Line

#### ***ETH Fixed Burst***

`ETH_FIXEDBURST_ENABLE`

`ETH_FIXEDBURST_DISABLE`

***ETH Flush Received Frame***

ETH\_FLUSHRECEIVEDFRAME\_ENABLE

ETH\_FLUSHRECEIVEDFRAME\_DISABLE

***ETH Forward Error Frames***

ETH\_FORWARDERRORFRAMES\_ENABLE

ETH\_FORWARDERRORFRAMES\_DISABLE

***ETH Forward Undersized Good Frames***

ETH\_FORWARDUNDERSIZEDGOODFRAMES\_ENABLE

ETH\_FORWARDUNDERSIZEDGOODFRAMES\_DISABLE

***ETH Inter Frame Gap***

ETH\_INTERFRAMEGAP\_96BIT minimum IFG between frames during transmission is 96Bit

ETH\_INTERFRAMEGAP\_88BIT minimum IFG between frames during transmission is 88Bit

ETH\_INTERFRAMEGAP\_80BIT minimum IFG between frames during transmission is 80Bit

ETH\_INTERFRAMEGAP\_72BIT minimum IFG between frames during transmission is 72Bit

ETH\_INTERFRAMEGAP\_64BIT minimum IFG between frames during transmission is 64Bit

ETH\_INTERFRAMEGAP\_56BIT minimum IFG between frames during transmission is 56Bit

ETH\_INTERFRAMEGAP\_48BIT minimum IFG between frames during transmission is 48Bit

ETH\_INTERFRAMEGAP\_40BIT minimum IFG between frames during transmission is 40Bit

***ETH Jabber***

ETH\_JABBER\_ENABLE

ETH\_JABBER\_DISABLE

***ETH Loop Back Mode***

ETH\_LOOPBACKMODE\_ENABLE

ETH\_LOOPBACKMODE\_DISABLE

***ETH MAC addresses***

ETH\_MAC\_ADDRESS0

ETH\_MAC\_ADDRESS1

ETH\_MAC\_ADDRESS2

ETH\_MAC\_ADDRESS3

***ETH MAC addresses filter Mask bytes***

ETH\_MAC\_ADDRESSMASK\_BYTE6 Mask MAC Address high reg bits [15:8]

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

***ETH MAC addresses filter SA DA***

ETH\_MAC\_ADDRESSFILTER\_SA  
ETH\_MAC\_ADDRESSFILTER\_DA

***ETH MAC Debug flags***

ETH\_MAC\_TXFIFO\_FULL  
ETH\_MAC\_TXFIFONOT\_EMPTY  
ETH\_MAC\_TXFIFO\_WRITE\_ACTIVE  
ETH\_MAC\_TXFIFO\_IDLE  
ETH\_MAC\_TXFIFO\_READ  
ETH\_MAC\_TXFIFO\_WAITING  
ETH\_MAC\_TXFIFO\_WRITING  
ETH\_MAC\_TRANSMISSION\_PAUSE  
ETH\_MAC\_TRANSMITFRAMECONTROLLER\_IDLE  
ETH\_MAC\_TRANSMITFRAMECONTROLLER\_WAITING  
ETH\_MAC\_TRANSMITFRAMECONTROLLER\_GENRATING\_PCF  
ETH\_MAC\_TRANSMITFRAMECONTROLLER\_TRANSFERRING  
ETH\_MAC\_MII\_TRANSMIT\_ACTIVE  
ETH\_MAC\_RXFIFO\_EMPTY  
ETH\_MAC\_RXFIFO\_BELOW\_THRESHOLD  
ETH\_MAC\_RXFIFO\_ABOVE\_THRESHOLD  
ETH\_MAC\_RXFIFO\_FULL  
ETH\_MAC\_READCONTROLLER\_IDLE  
ETH\_MAC\_READCONTROLLER\_READING\_DATA  
ETH\_MAC\_READCONTROLLER\_READING\_STATUS  
ETH\_MAC\_READCONTROLLER\_FLUSHING  
ETH\_MAC\_RXFIFO\_WRITE\_ACTIVE  
ETH\_MAC\_SMALL\_FIFO\_NOTACTIVE  
ETH\_MAC\_SMALL\_FIFO\_READ\_ACTIVE  
ETH\_MAC\_SMALL\_FIFO\_WRITE\_ACTIVE  
ETH\_MAC\_SMALL\_FIFO\_RW\_ACTIVE  
ETH\_MAC\_MII\_RECEIVE\_PROTOCOL\_ACTIVE

***ETH MAC Flags***

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

#### ***ETH MAC Interrupts***

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

#### ***ETH Media Interface***

ETH_MEDIA_INTERFACE_MII	
ETH_MEDIA_INTERFACE_RMII	

#### ***ETH MMC Rx Interrupts***

ETH_MMCI_T_RGUF	When Rx good unicast frames counter reaches half the maximum value
ETH_MMCI_T_RFAE	When Rx alignment error counter reaches half the maximum value
ETH_MMCI_T_RFCE	When Rx crc error counter reaches half the maximum value

#### ***ETH MMC Tx Interrupts***

ETH_MMCI_T_TGF	When Tx good frame counter reaches half the maximum value
ETH_MMCI_T_TGFMSC	When Tx good multi col counter reaches half the maximum value
ETH_MMCI_T_TGFSC	When Tx good single col counter reaches half the maximum value

#### ***ETH Multicast Frames Filter***

ETH_MULTICASTFRAMESFILTER_PERFECTHASHTABLE	
ETH_MULTICASTFRAMESFILTER_HASHTABLE	
ETH_MULTICASTFRAMESFILTER_PERFECT	
ETH_MULTICASTFRAMESFILTER_NONE	

#### ***ETH Pass Control Frames***

ETH_PASSCONTROLFRAMES_BLOCKALL	MAC filters all control frames from reaching the application
ETH_PASSCONTROLFRAMES_FORWARDALL	MAC forwards all control frames to application even if they fail the Address Filter

<code>ETH_PASSCONTROLFRAMES_FORWARDPASSEDADDRFILTER</code>	MAC forwards control frames that pass the Address Filter.
--	---

***ETH Pause Low Threshold***

<code>ETH_PAUSELOWTHRESHOLD_MINUS4</code>	Pause time minus 4 slot times
<code>ETH_PAUSELOWTHRESHOLD_MINUS28</code>	Pause time minus 28 slot times
<code>ETH_PAUSELOWTHRESHOLD_MINUS144</code>	Pause time minus 144 slot times
<code>ETH_PAUSELOWTHRESHOLD_MINUS256</code>	Pause time minus 256 slot times

***ETH PMT Flags***

<code>ETH_PMT_FLAG_WUFFRPR</code>	Wake-Up Frame Filter Register Pointer Reset
<code>ETH_PMT_FLAG_WUFR</code>	Wake-Up Frame Received
<code>ETH_PMT_FLAG_MPR</code>	Magic Packet Received

***ETH Private Constants***

<code>LINKED_STATE_TIMEOUT_VALUE</code>
<code>AUTONEGO_COMPLETED_TIMEOUT_VALUE</code>

***ETH Private Defines***

<code>ETH_REG_WRITE_DELAY</code>
<code>ETH_SUCCESS</code>
<code>ETH_ERROR</code>
<code>ETH_DMATXDESC_COLLISION_COUNTSHIFT</code>
<code>ETH_DMATXDESC_BUFFER2_SIZESSHIFT</code>
<code>ETH_DMARXDESC_FRAME_LENGTHSHIFT</code>
<code>ETH_DMARXDESC_BUFFER2_SIZESSHIFT</code>
<code>ETH_DMARXDESC_FRAMELENGTHSHIFT</code>
<code>ETH_MAC_ADDR_HBASE</code>
<code>ETH_MAC_ADDR_LBASE</code>
<code>ETH_MACMIIAR_CR_MASK</code>
<code>ETH_MACCR_CLEAR_MASK</code>
<code>ETH_MACFCR_CLEAR_MASK</code>
<code>ETH_DMAOMR_CLEAR_MASK</code>
<code>ETH_WAKEUP_REGISTER_LENGTH</code>
<code>ETH_DMA_RX_OVERFLOW_MISSEDFRAMES_COUNTERSHIFT</code>

***ETH Private Macros***

<code>IS_ETH_PHY_ADDRESS</code>
<code>IS_ETH_AUTONEGOTIATION</code>
<code>IS_ETH_SPEED</code>

IS\_ETH\_DUPLEX\_MODE  
IS\_ETH\_RX\_MODE  
IS\_ETH\_CHECKSUM\_MODE  
IS\_ETH\_MEDIA\_INTERFACE  
IS\_ETH\_WATCHDOG  
IS\_ETH\_JABBER  
IS\_ETH\_INTER\_FRAME\_GAP  
IS\_ETH\_CARRIER\_SENSE  
IS\_ETH\_RECEIVE\_OWN  
IS\_ETH\_LOOPBACK\_MODE  
IS\_ETH\_CHECKSUM\_OFFLOAD  
IS\_ETH\_RETRY\_TRANSMISSION  
IS\_ETH\_AUTOMATIC\_PADCRC\_STRIP  
IS\_ETH\_BACKOFF\_LIMIT  
IS\_ETH\_DEFERRAL\_CHECK  
IS\_ETH\_RECEIVE\_ALL  
IS\_ETH\_SOURCE\_ADDR\_FILTER  
IS\_ETH\_CONTROL\_FRAMES  
IS\_ETH\_BROADCAST\_FRAMES\_RECEPTION  
IS\_ETH\_DESTINATION\_ADDR\_FILTER  
IS\_ETH\_PROMISCUOUS\_MODE  
IS\_ETH\_MULTICAST\_FRAMES\_FILTER  
IS\_ETH\_UNICAST\_FRAMES\_FILTER  
IS\_ETH\_PAUSE\_TIME  
IS\_ETH\_ZEROQUANTA\_PAUSE  
IS\_ETH\_PAUSE\_LOW\_THRESHOLD  
IS\_ETH\_UNICAST\_PAUSE\_FRAME\_DETECT  
IS\_ETH\_RECEIVE\_FLOWCONTROL  
IS\_ETH\_TRANSMIT\_FLOWCONTROL  
IS\_ETH\_VLAN\_TAG\_COMPARISON  
IS\_ETH\_VLAN\_TAG\_IDENTIFIER  
IS\_ETH\_MAC\_ADDRESS0123  
IS\_ETH\_MAC\_ADDRESS123  
IS\_ETH\_MAC\_ADDRESS\_FILTER  
IS\_ETH\_MAC\_ADDRESS\_MASK  
IS\_ETH\_DROP\_TCPIP\_CHECKSUM\_FRAME

IS\_ETH\_RECEIVE\_STORE\_FORWARD  
IS\_ETH\_FLUSH\_RECEIVE\_FRAME  
IS\_ETH\_TRANSMIT\_STORE\_FORWARD  
IS\_ETH\_TRANSMIT\_THRESHOLD\_CONTROL  
IS\_ETH\_FORWARD\_ERROR\_FRAMES  
IS\_ETH\_FORWARD\_UNDERSIZED\_GOOD\_FRAMES  
IS\_ETH\_RECEIVE\_THRESHOLD\_CONTROL  
IS\_ETH\_SECOND\_FRAME\_OPERATE  
IS\_ETH\_ADDRESS\_ALIGNED\_BEATS  
IS\_ETH\_FIXED\_BURST  
IS\_ETH\_RXDMA\_BURST\_LENGTH  
IS\_ETH\_TXDMA\_BURST\_LENGTH  
IS\_ETH\_DMA\_DESC\_SKIP\_LENGTH  
IS\_ETH\_DMA\_ARBITRATION\_ROUNDROBIN\_RXTX  
IS\_ETH\_DMATXDESC\_GET\_FLAG  
IS\_ETH\_DMA\_TXDESC\_SEGMENT  
IS\_ETH\_DMA\_TXDESC\_CHECKSUM  
IS\_ETH\_DMATXDESC\_BUFFER\_SIZE  
IS\_ETH\_DMARXDESC\_GET\_FLAG  
IS\_ETH\_DMA\_RXDESC\_BUFFER  
IS\_ETH\_PMT\_GET\_FLAG  
IS\_ETH\_DMA\_FLAG  
IS\_ETH\_DMA\_GET\_FLAG  
IS\_ETH\_MAC\_IT  
IS\_ETH\_MAC\_GET\_IT  
IS\_ETH\_MAC\_GET\_FLAG  
IS\_ETH\_DMA\_IT  
IS\_ETH\_DMA\_GET\_IT  
IS\_ETH\_DMA\_GET\_OVERFLOW  
IS\_ETH\_MMC\_IT  
IS\_ETH\_MMC\_GET\_IT  
IS\_ETH\_ENHANCED\_DESCRIPTOR\_FORMAT  
**ETH Promiscuous Mode**  
ETH\_PROMISCUOUS\_MODE\_ENABLE  
ETH\_PROMISCUOUS\_MODE\_DISABLE  
**ETH Receive All**

ETH_RECEIVEALL_ENABLE	
ETH_RECEIVEALL_DISABLE	
<b>ETH Receive Flow Control</b>	
ETH_RECEIVEFLOWCONTROL_ENABLE	
ETH_RECEIVEFLOWCONTROL_DISABLE	
<b>ETH Receive Own</b>	
ETH_RECEIVEOWN_ENABLE	
ETH_RECEIVEOWN_DISABLE	
<b>ETH Receive Store Forward</b>	
ETH_RECEIVESTOREFORWARD_ENABLE	
ETH_RECEIVESTOREFORWARD_DISABLE	
<b>ETH Receive Threshold Control</b>	
ETH_RECEIVEDTHRESHOLDCONTROL_64BYTES	threshold level of the MTL Receive FIFO is 64 Bytes
ETH_RECEIVEDTHRESHOLDCONTROL_32BYTES	threshold level of the MTL Receive FIFO is 32 Bytes
ETH_RECEIVEDTHRESHOLDCONTROL_96BYTES	threshold level of the MTL Receive FIFO is 96 Bytes
ETH_RECEIVEDTHRESHOLDCONTROL_128BYTES	threshold level of the MTL Receive FIFO is 128 Bytes
<b>ETH Retry Transmission</b>	
ETH_RETRYTRANSMISSION_ENABLE	
ETH_RETRYTRANSMISSION_DISABLE	
<b>ETH Rx DMA Burst Length</b>	
ETH_RXDMABURSTLENGTH_1BEAT	maximum number of beats to be transferred in one RxDMA transaction is 1
ETH_RXDMABURSTLENGTH_2BEAT	maximum number of beats to be transferred in one RxDMA transaction is 2
ETH_RXDMABURSTLENGTH_4BEAT	maximum number of beats to be transferred in one RxDMA transaction is 4
ETH_RXDMABURSTLENGTH_8BEAT	maximum number of beats to be transferred in one RxDMA transaction is 8
ETH_RXDMABURSTLENGTH_16BEAT	maximum number of beats to be transferred in one RxDMA transaction is 16
ETH_RXDMABURSTLENGTH_32BEAT	maximum number of beats to be transferred in one RxDMA transaction is 32

ETH_RXDMABURSTLENGTH_4XPBL_4BEAT	maximum number of beats to be transferred in one RxDMA transaction is 4
ETH_RXDMABURSTLENGTH_4XPBL_8BEAT	maximum number of beats to be transferred in one RxDMA transaction is 8
ETH_RXDMABURSTLENGTH_4XPBL_16BEAT	maximum number of beats to be transferred in one RxDMA transaction is 16
ETH_RXDMABURSTLENGTH_4XPBL_32BEAT	maximum number of beats to be transferred in one RxDMA transaction is 32
ETH_RXDMABURSTLENGTH_4XPBL_64BEAT	maximum number of beats to be transferred in one RxDMA transaction is 64
ETH_RXDMABURSTLENGTH_4XPBL_128BEAT	maximum number of beats to be transferred in one RxDMA transaction is 128

***ETH Rx Mode***

ETH\_RXPOLLING\_MODE

ETH\_RXINTERRUPT\_MODE

***ETH Second Frame Operate***

ETH\_SECONDFRAMEOPERARTE\_ENABLE

ETH\_SECONDFRAMEOPERARTE\_DISABLE

***ETH Source Addr Filter***

ETH\_SOURCEADDRFILTER\_NORMAL\_ENABLE

ETH\_SOURCEADDRFILTER\_INVERSE\_ENABLE

ETH\_SOURCEADDRFILTER\_DISABLE

***ETH Speed***

ETH\_SPEED\_10M

ETH\_SPEED\_100M

***ETH Transmit Flow Control***

ETH\_TRANSMITFLOWCONTROL\_ENABLE

ETH\_TRANSMITFLOWCONTROL\_DISABLE

***ETH Transmit Store Forward***

ETH\_TRANSMITSTOREFORWARD\_ENABLE

ETH\_TRANSMITSTOREFORWARD\_DISABLE

***ETH Transmit Threshold Control***ETH\_TRANSMITTHRESHOLDCONTROL\_64BYTES threshold level of the MTL  
Transmit FIFO is 64 Bytes

ETH\_TRANSMITTHRESHOLDCONTROL\_128BYTES threshold level of the MTL

ETH_TRANSMITTHRESHOLDCONTROL_192BYTES	Transmit FIFO is 128 Bytes threshold level of the MTL Transmit FIFO is 192 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_256BYTES	threshold level of the MTL Transmit FIFO is 256 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_40BYTES	threshold level of the MTL Transmit FIFO is 40 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_32BYTES	threshold level of the MTL Transmit FIFO is 32 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_24BYTES	threshold level of the MTL Transmit FIFO is 24 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_16BYTES	threshold level of the MTL Transmit FIFO is 16 Bytes
<b>ETH Tx DMA Burst Length</b>	
ETH_TXDMABURSTLENGTH_1BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 1
ETH_TXDMABURSTLENGTH_2BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 2
ETH_TXDMABURSTLENGTH_4BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 4
ETH_TXDMABURSTLENGTH_8BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 8
ETH_TXDMABURSTLENGTH_16BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 16
ETH_TXDMABURSTLENGTH_32BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 32
ETH_TXDMABURSTLENGTH_4XPBL_4BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 4
ETH_TXDMABURSTLENGTH_4XPBL_8BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 8
ETH_TXDMABURSTLENGTH_4XPBL_16BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 16
ETH_TXDMABURSTLENGTH_4XPBL_32BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 32
ETH_TXDMABURSTLENGTH_4XPBL_64BEAT	maximum number of beats to be transferred in one TxDMA (or both)

transaction is 64

ETH\_TXDMABURSTLENGTH\_4XPBL\_128BEAT maximum number of beats to be transferred in one TxDMA (or both) transaction is 128

***ETH Unicast Frames Filter***

ETH\_UNICASTFRAMESFILTER\_PERFECTHASHTABLE

ETH\_UNICASTFRAMESFILTER\_HASHTABLE

ETH\_UNICASTFRAMESFILTER\_PERFECT

***ETH Unicast Pause Frame Detect***

ETH\_UNICASTPAUSEFRAMEDETECT\_ENABLE

ETH\_UNICASTPAUSEFRAMEDETECT\_DISABLE

***ETH VLAN Tag Comparison***

ETH\_VLANTAGCOMPARISON\_12BIT

ETH\_VLANTAGCOMPARISON\_16BIT

***ETH Watchdog***

ETH\_WATCHDOG\_ENABLE

ETH\_WATCHDOG\_DISABLE

***ETH Zero Quanta Pause***

ETH\_ZEROQUANTAPAUSE\_ENABLE

ETH\_ZEROQUANTAPAUSE\_DISABLE

## 16 HAL FLASH Generic Driver

### 16.1 FLASH Firmware driver registers structures

#### 16.1.1 FLASH\_ProcessTypeDef

##### 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 uint32_t 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 uint32_t FLASH_ProcessTypeDef::ErrorCode`

### 16.2 FLASH Firmware driver API description

#### 16.2.1 FLASH peripheral features

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

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

## 16.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32F2xx 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

## 16.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

This section contains the following APIs:

- [`HAL\_FLASH\_Program\(\)`](#)
- [`HAL\_FLASH\_Program\_IT\(\)`](#)
- [`HAL\_FLASH\_IRQHandler\(\)`](#)
- [`HAL\_FLASH\_EndOfOperationCallback\(\)`](#)
- [`HAL\_FLASH\_OperationErrorCallback\(\)`](#)

## 16.2.4 Peripheral Control functions

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

This section contains the following APIs:

- [`HAL\_FLASH\_Unlock\(\)`](#)
- [`HAL\_FLASH\_Lock\(\)`](#)
- [`HAL\_FLASH\_OB\_Unlock\(\)`](#)
- [`HAL\_FLASH\_OB\_Lock\(\)`](#)
- [`HAL\_FLASH\_OB\_Launch\(\)`](#)

## 16.2.5 Peripheral Errors functions

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

This section contains the following APIs:

- [`HAL\_FLASH\_GetError\(\)`](#)

- *[FLASH\\_WaitForLastOperation\(\)](#)*

### 16.2.6 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 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	• HAL_StatusTypeDef HAL Status

### 16.2.7 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	• HAL Status

### 16.2.8 HAL\_FLASH\_IRQHandler

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

### 16.2.9 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>

### 16.2.10 HAL\_FLASH\_OperationErrorHandler

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>

### 16.2.11 HAL\_FLASH\_Unlock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Unlock (void )</b>
Function Description	Unlock the FLASH control register access.
Return values	<ul style="list-style-type: none"> <li>• HAL Status</li> </ul>

### 16.2.12 HAL\_FLASH\_Lock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_Lock (void )</b>
Function Description	Locks the FLASH control register access.
Return values	<ul style="list-style-type: none"> <li>• HAL Status</li> </ul>

### 16.2.13 HAL\_FLASH\_OB\_Unlock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void )</b>
Function Description	Unlock the FLASH Option Control Registers access.
Return values	<ul style="list-style-type: none"> <li>• HAL Status</li> </ul>

### 16.2.14 HAL\_FLASH\_OB\_Lock

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Lock (void )</b>
Function Description	Lock the FLASH Option Control Registers access.
Return values	<ul style="list-style-type: none"> <li>• HAL Status</li> </ul>

### 16.2.15 HAL\_FLASH\_OB\_Launch

Function Name	<b>HAL_StatusTypeDef HAL_FLASH_OB_Launch (void )</b>
Function Description	Launch the option byte loading.
Return values	<ul style="list-style-type: none"> <li>• HAL Status</li> </ul>

### 16.2.16 HAL\_FLASH\_GetError

Function Name	<b>uint32_t HAL_FLASH_GetError (void )</b>
Function Description	Get the specific FLASH error flag.
Return values	<ul style="list-style-type: none"> <li>• FLASH_ErrorCode The returned value can be a combination of: HAL_FLASH_ERROR_NONE: FLASH Programming Sequence error flag HAL_FLASH_ERROR_PGS: FLASH</li> </ul>

Programming Sequence error flag  
 HAL\_FLASH\_ERROR\_PGP: FLASH Programming  
 Parallelism error flag HAL\_FLASH\_ERROR\_PGA: FLASH  
 Programming Alignment error flag  
 HAL\_FLASH\_ERROR\_WRP: FLASH Write protected error  
 flag HAL\_FLASH\_ERROR\_OPERATION: FLASH operation  
 Error flag

### 16.2.17 **FLASH\_WaitForLastOperation**

Function Name	<b>HAL_StatusTypeDef</b> <b>FLASH_WaitForLastOperation</b> ( <b>uint32_t</b> <b>Timeout</b> )
Function Description	Wait for a FLASH operation to complete.
Parameters	<ul style="list-style-type: none"> <li>• <b>Timeout:</b> maximum flash operation timeout</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL Status</li> </ul>

## 16.3 **FLASH Firmware driver defines**

### 16.3.1 **FLASH**

#### ***FLASH Error Code***

<b>HAL_FLASH_ERROR_NONE</b>	No error
<b>HAL_FLASH_ERROR_PGS</b>	Programming Sequence error
<b>HAL_FLASH_ERROR_PGP</b>	Programming Parallelism error
<b>HAL_FLASH_ERROR_PGA</b>	Programming Alignment error
<b>HAL_FLASH_ERROR_WRP</b>	Write protection error
<b>HAL_FLASH_ERROR_OPERATION</b>	Operation Error

#### ***FLASH Exported Macros***

<b>_HAL_FLASH_SET_LATENCY</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>• Set the FLASH Latency.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <b>_LATENCY_</b>: FLASH Latency The value of this parameter depend on device used within the same series</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• none</li> </ul>
<b>_HAL_FLASH_GET_LATENCY</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>• Get the FLASH Latency.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• FLASH: Latency The value of this parameter depend on device used within the same series</li> </ul>

`__HAL_FLASH_PREFETCH_BUFFER_ENABLE`

**Description:**

- Enable the FLASH prefetch buffer.

**Return value:**

- none

`__HAL_FLASH_PREFETCH_BUFFER_DISABLE`

**Description:**

- Disable the FLASH prefetch buffer.

**Return value:**

- none

`__HAL_FLASH_INSTRUCTION_CACHE_ENABLE`

**Description:**

- Enable the FLASH instruction cache.

**Return value:**

- none

`__HAL_FLASH_INSTRUCTION_CACHE_DISABLE`

**Description:**

- Disable the FLASH instruction cache.

**Return value:**

- none

`__HAL_FLASH_DATA_CACHE_ENABLE`

**Description:**

- Enable the FLASH data cache.

**Return value:**

- none

`__HAL_FLASH_DATA_CACHE_DISABLE`

**Description:**

- Disable the FLASH data cache.

**Return value:**

- none

`__HAL_FLASH_INSTRUCTION_CACHE_RESET`

**Description:**

- Resets the FLASH instruction Cache.

**Return value:**

- None

**Notes:**

- This function must be used only when the Instruction Cache is disabled.

`__HAL_FLASH_DATA_CACHE_RESET`

**Description:**

- Resets the FLASH data Cache.

**Return value:**

- None

**Notes:**

- This function must be used only when the data Cache is disabled.

`__HAL_FLASH_ENABLE_IT`

**Description:**

- Enable the specified FLASH interrupt.

**Parameters:**

- `__INTERRUPT__`: : FLASH interrupt This parameter can be any combination of the following values:
  - `FLASH_IT_EOP`: End of FLASH Operation Interrupt
  - `FLASH_IT_ERR`: Error Interrupt

**Return value:**

- none

`__HAL_FLASH_DISABLE_IT`

**Description:**

- Disable the specified FLASH interrupt.

**Parameters:**

- `__INTERRUPT__`: : FLASH interrupt This parameter can be any combination of the following values:
  - `FLASH_IT_EOP`: End of FLASH Operation Interrupt
  - `FLASH_IT_ERR`: Error Interrupt

**Return value:**

- none

`__HAL_FLASH_GET_FLAG`

**Description:**

- Get the specified FLASH flag status.

**Parameters:**

- `__FLAG__`: specifies the FLASH flag to check. This parameter can be one of the following values:
  - `FLASH_FLAG_EOP` : FLASH End of Operation flag
  - `FLASH_FLAG_OPERR` :

- FLASH operation Error flag
- FLASH\_FLAG\_WRPERR: FLASH Write protected error flag
- FLASH\_FLAG\_PGAERR: FLASH Programming Alignment error flag
- FLASH\_FLAG\_PGPERR: FLASH Programming Parallelism error flag
- FLASH\_FLAG\_PGSERR: FLASH Programming Sequence error flag
- FLASH\_FLAG\_BSY : FLASH Busy flag

**Return value:**

- The new state of \_\_FLAG\_\_ (SET or RESET).

**\_HAL\_FLASH\_CLEAR\_FLAG****Description:**

- Clear the specified FLASH flag.

**Parameters:**

- \_\_FLAG\_\_: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
  - FLASH\_FLAG\_EOP : FLASH End of Operation flag
  - FLASH\_FLAG\_OPERR : FLASH operation Error flag
  - FLASH\_FLAG\_WRPERR: FLASH Write protected error flag
  - FLASH\_FLAG\_PGAERR: FLASH Programming Alignment error flag
  - FLASH\_FLAG\_PGPERR: FLASH Programming Parallelism error flag
  - FLASH\_FLAG\_PGSERR: FLASH Programming Sequence error flag

**Return value:**

- none

***FLASH Flag definition***

FLASH_FLAG_EOP	FLASH End of Operation flag
FLASH_FLAG_OPERR	FLASH operation Error flag
FLASH_FLAG_WRPERR	FLASH Write protected error flag

FLASH\_FLAG\_PGAERR    FLASH Programming Alignment error flag  
FLASH\_FLAG\_PGPERR    FLASH Programming Parallelism error flag  
FLASH\_FLAG\_PGSERR    FLASH Programming Sequence error flag  
FLASH\_FLAG\_BSY        FLASH Busy flag

***FLASH Interrupt definition***

FLASH\_IT\_EOP        End of FLASH Operation Interrupt source  
FLASH\_IT\_ERR        Error Interrupt source

***FLASH Private macros to check input parameters***

IS\_FLASH\_TYPEPROGRAM

***FLASH Keys***

RDP\_KEY  
FLASH\_KEY1  
FLASH\_KEY2  
FLASH\_OPT\_KEY1  
FLASH\_OPT\_KEY2

***FLASH Latency***

FLASH\_LATENCY\_0    FLASH Zero Latency cycle  
FLASH\_LATENCY\_1    FLASH One Latency cycle  
FLASH\_LATENCY\_2    FLASH Two Latency cycles  
FLASH\_LATENCY\_3    FLASH Three Latency cycles  
FLASH\_LATENCY\_4    FLASH Four Latency cycles  
FLASH\_LATENCY\_5    FLASH Five Latency cycles  
FLASH\_LATENCY\_6    FLASH Six Latency cycles  
FLASH\_LATENCY\_7    FLASH Seven Latency cycles

***FLASH Private Constants***

FLASH\_TIMEOUT\_VALUE  
ACR\_BYTE0\_ADDRESS  
OPTCR\_BYTE0\_ADDRESS  
OPTCR\_BYTE1\_ADDRESS  
OPTCR\_BYTE2\_ADDRESS  
OPTCR\_BYTE3\_ADDRESS

***FLASH Program Parallelism***

FLASH\_PSIZE\_BYTE  
FLASH\_PSIZE\_HALF\_WORD  
FLASH\_PSIZE\_WORD  
FLASH\_PSIZE\_DOUBLE\_WORD

**CR\_PSIZE\_MASK**

***FLASH Type Program***

<b>FLASH_TYPEPROGRAM_BYTE</b>	Program byte (8-bit) at a specified address
<b>FLASH_TYPEPROGRAM_HALFWORD</b>	Program a half-word (16-bit) at a specified address
<b>FLASH_TYPEPROGRAM_WORD</b>	Program a word (32-bit) at a specified address
<b>FLASH_TYPEPROGRAM_DOUBLEWORD</b>	Program a double word (64-bit) at a specified address

## 17 HAL FLASH Extension Driver

### 17.1 FLASHEx Firmware driver registers structures

#### 17.1.1 FLASH\_EraseInitTypeDef

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

#### 17.1.2 FLASH\_OBProgramInitTypeDef

##### 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***  
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.

## 17.2 FLASHEx Firmware driver API description

### 17.2.1 Flash Extension features

### 17.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32F2xx 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

### 17.2.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extension FLASH programming operations.

This section contains the following APIs:

- ***HAL\_FLASHEx\_Erase()***
- ***HAL\_FLASHEx\_Erase\_IT()***
- ***HAL\_FLASHEx\_OBProgram()***
- ***HAL\_FLASHEx\_OBGetConfig()***

### 17.2.4 HAL\_FLASHEx\_Erase

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraselTypeDef * pEraselInit, uint32_t * SectorError)</b>
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 FLASH_EraselTypeDef 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>HAL Status</li> </ul>

### 17.2.5 HAL\_FLASHEx\_Erase\_IT

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraselTypeDef * pEraselInit)</b>
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>pEraselInit:</b> pointer to an FLASH_EraselTypeDef structure that contains the configuration information for the erasing.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL Status</li> </ul>

### 17.2.6 HAL\_FLASHEx\_OBProgram

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)</b>
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> <li><b>pOBInit:</b> pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL Status</li> </ul>

### 17.2.7 HAL\_FLASHEx\_OBGetConfig

Function Name	<b>void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)</b>
Function Description	Get the Option byte configuration.
Parameters	<ul style="list-style-type: none"> <li><b>pOBInit:</b> pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

## 17.3 FLASHEx Firmware driver defines

### 17.3.1 FLASHEx

*FLASH Banks*

**FLASH\_BANK\_1**      Bank 1

*FLASH BOR Reset Level*



---

OB_BOR_LEVEL3	Supply voltage ranges from 2.70 to 3.60 V
OB_BOR_LEVEL2	Supply voltage ranges from 2.40 to 2.70 V
OB_BOR_LEVEL1	Supply voltage ranges from 2.10 to 2.40 V
OB_BOR_OFF	Supply voltage ranges from 1.62 to 2.10 V

***FLASH Private macros to check input parameters***

IS\_FLASH\_TYPEERASE  
IS\_VOLTAGERANGE  
IS\_WRPSTATE  
IS\_OPTIONBYTE  
IS\_OB\_RDP\_LEVEL  
IS\_OB\_IWDG\_SOURCE  
IS\_OB\_STOP\_SOURCE  
IS\_OB\_STDBY\_SOURCE  
IS\_OB\_BOR\_LEVEL  
IS\_FLASH\_LATENCY  
IS\_FLASH\_BANK  
IS\_FLASH\_SECTOR  
IS\_FLASH\_ADDRESS  
IS\_FLASH\_NBSECTORS  
IS\_OB\_WRP\_SECTOR

***FLASH Mass Erase bit***

FLASH\_MER\_BIT      only 1 MER Bit

***FLASH Option Bytes IWatchdog***

OB\_IWDG\_SW      Software IWDG selected  
OB\_IWDG\_HW      Hardware IWDG selected

***FLASH Option Bytes nRST\_STDBY***

OB\_STDBY\_NO\_RST    No reset generated when entering in STANDBY  
OB\_STDBY\_RST      Reset generated when entering in STANDBY

***FLASH Option Bytes nRST\_STOP***

OB\_STOP\_NO\_RST    No reset generated when entering in STOP  
OB\_STOP\_RST      Reset generated when entering in STOP

***FLASH Option Bytes Read Protection***

OB\_RDP\_LEVEL\_0  
OB\_RDP\_LEVEL\_1  
OB\_RDP\_LEVEL\_2    Warning: When enabling read protection level 2 it's no more possible to go back to level 1 or 0

***FLASH Option Bytes Write Protection***

---

OB_WRP_SECTOR_0	Write protection of Sector0
OB_WRP_SECTOR_1	Write protection of Sector1
OB_WRP_SECTOR_2	Write protection of Sector2
OB_WRP_SECTOR_3	Write protection of Sector3
OB_WRP_SECTOR_4	Write protection of Sector4
OB_WRP_SECTOR_5	Write protection of Sector5
OB_WRP_SECTOR_6	Write protection of Sector6
OB_WRP_SECTOR_7	Write protection of Sector7
OB_WRP_SECTOR_8	Write protection of Sector8
OB_WRP_SECTOR_9	Write protection of Sector9
OB_WRP_SECTOR_10	Write protection of Sector10
OB_WRP_SECTOR_11	Write protection of Sector11
OB_WRP_SECTOR_All	Write protection of all Sectors

***FLASH Option Type***

OPTIONBYTE_WRP	WRP option byte configuration
OPTIONBYTE_RDP	RDP option byte configuration
OPTIONBYTE_USER	USER option byte configuration
OPTIONBYTE_BOR	BOR option byte configuration

***FLASH Private Constants***

FLASH\_TIMEOUT\_VALUE

FLASH\_SECTOR\_TOTAL

***FLASH Sectors***

FLASH_SECTOR_0	Sector Number 0
FLASH_SECTOR_1	Sector Number 1
FLASH_SECTOR_2	Sector Number 2
FLASH_SECTOR_3	Sector Number 3
FLASH_SECTOR_4	Sector Number 4
FLASH_SECTOR_5	Sector Number 5
FLASH_SECTOR_6	Sector Number 6
FLASH_SECTOR_7	Sector Number 7
FLASH_SECTOR_8	Sector Number 8
FLASH_SECTOR_9	Sector Number 9
FLASH_SECTOR_10	Sector Number 10
FLASH_SECTOR_11	Sector Number 11

***FLASH Type Erase***

FLASH\_TYPEERASE\_SECTORS      Sectors erase only

**FLASH\_TYPEERASE\_MASSERASE** Flash Mass erase activation

***FLASH Voltage Range***

**FLASH\_VOLTAGE\_RANGE\_1** Device operating range: 1.8V to 2.1V

**FLASH\_VOLTAGE\_RANGE\_2** Device operating range: 2.1V to 2.7V

**FLASH\_VOLTAGE\_RANGE\_3** Device operating range: 2.7V to 3.6V

**FLASH\_VOLTAGE\_RANGE\_4** Device operating range: 2.7V to 3.6V + External V<sub>pp</sub>

***FLASH WRP State***

**OB\_WRPSTATE\_DISABLE** Disable the write protection of the desired bank 1 sectors

**OB\_WRPSTATE\_ENABLE** Enable the write protection of the desired bank 1 sectors

## 18 HAL GPIO Generic Driver

### 18.1 GPIO Firmware driver registers structures

#### 18.1.1 GPIO\_InitTypeDef

##### Data Fields

- *uint32\_t Pin*
- *uint32\_t Mode*
- *uint32\_t Pull*
- *uint32\_t Speed*
- *uint32\_t Alternate*

##### Field Documentation

- ***uint32\_t GPIO\_InitTypeDef::Pin***  
Specifies the GPIO pins to be configured. This parameter can be any value of [\*GPIO\\_pins\\_define\*](#)
- ***uint32\_t GPIO\_InitTypeDef::Mode***  
Specifies the operating mode for the selected pins. This parameter can be a value of [\*GPIO\\_mode\\_define\*](#)
- ***uint32\_t GPIO\_InitTypeDef::Pull***  
Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of [\*GPIO\\_pull\\_define\*](#)
- ***uint32\_t GPIO\_InitTypeDef::Speed***  
Specifies the speed for the selected pins. This parameter can be a value of [\*GPIO\\_speed\\_define\*](#)
- ***uint32\_t GPIO\_InitTypeDef::Alternate***  
Peripheral to be connected to the selected pins. This parameter can be a value of [\*GPIO\\_Alternate\\_function\\_selection\*](#)

### 18.2 GPIO Firmware driver API description

#### 18.2.1 GPIO Peripheral features

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes:

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

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

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

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

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

The external interrupt/event controller consists of up to 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.

## 18.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function:  
`__HAL_RCC_GPIOx_CLK_ENABLE()`.
2. Configure the GPIO pin(s) using `HAL_GPIO_Init()`.
  - Configure the IO mode using "Mode" member from `GPIO_InitTypeDef` structure
  - Activate Pull-up, Pull-down resistor using "Pull" member from `GPIO_InitTypeDef` structure.
  - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from `GPIO_InitTypeDef` structure.
  - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from `GPIO_InitTypeDef` structure.
  - Analog mode is required when a pin is to be used as ADC channel or DAC output.
  - In case of external interrupt/event selection the "Mode" member from `GPIO_InitTypeDef` structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using `HAL_NVIC_SetPriority()` and enable it using `HAL_NVIC_EnableIRQ()`.
4. To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()`.
5. To set/reset the level of a pin configured in output mode use  
`HAL_GPIO_WritePin()`/`HAL_GPIO_TogglePin()`.
6. To lock pin configuration until next reset use `HAL_GPIO_LockPin()`.
7. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
8. The LSE oscillator pins OSC32\_IN and OSC32\_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
9. The HSE oscillator pins OSC\_IN/OSC\_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

## 18.2.3 Initialization and de-initialization functions

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

This section contains the following APIs:

- [`HAL\_GPIO\_Init\(\)`](#)
- [`HAL\_GPIO\_DeInit\(\)`](#)

### 18.2.4 IO operation functions

This section contains the following APIs:

- [`HAL\_GPIO\_ReadPin\(\)`](#)
- [`HAL\_GPIO\_WritePin\(\)`](#)
- [`HAL\_GPIO\_TogglePin\(\)`](#)
- [`HAL\_GPIO\_LockPin\(\)`](#)
- [`HAL\_GPIO\_EXTI\_IRQHandler\(\)`](#)
- [`HAL\_GPIO\_EXTI\_Callback\(\)`](#)

### 18.2.5 `HAL_GPIO_Init`

Function Name	<code>void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)</code>
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the <code>GPIO_Init</code> .
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b>: where x can be (A..I) to select the GPIO peripheral.</li> <li>• <b>GPIO_InitStruct</b>: pointer to a <code>GPIO_InitTypeDef</code> 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>

### 18.2.6 `HAL_GPIO_DeInit`

Function Name	<code>void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)</code>
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..I) to select the GPIO peripheral.</li> <li>• <b>GPIO_Pin</b>: specifies the port bit to be written. This parameter can be one of <code>GPIO_PIN_x</code> where x can be (0..15).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 18.2.7 `HAL_GPIO_ReadPin`

Function Name	<code>GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Reads the specified input port pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx</b>: where x can be (A..I) to select the GPIO peripheral.</li> <li>• <b>GPIO_Pin</b>: specifies the port bit to read. This parameter can be <code>GPIO_PIN_x</code> where x can be (0..15).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• The input port pin value.</li> </ul>

### 18.2.8 `HAL_GPIO_WritePin`

Function Name	<code>void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)</code>
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..I) to select the GPIO peripheral for all STM32F2XX 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_PIN_RESET: to clear the port pin GPIO_PIN_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>

### 18.2.9 HAL\_GPIO\_TogglePin

Function Name	<code>void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Toggles the specified GPIO pins.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx:</b> where x can be (A..I) to select the GPIO peripheral.</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>

### 18.2.10 HAL\_GPIO\_LockPin

Function Name	<code>HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Locks GPIO Pins configuration registers.
Parameters	<ul style="list-style-type: none"> <li><b>GPIOx:</b> where x can be (A..I) to select the GPIO peripheral for STM32F2XX family</li> <li><b>GPIO_Pin:</b> specifies the port bit to be locked. This parameter can be any combination of GPIO_PIN_x 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>The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.</li> <li>The configuration of the locked GPIO pins can no longer be modified until the next reset.</li> </ul>

### 18.2.11 HAL\_GPIO\_EXTI\_IRQHandler

Function Name	<code>void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)</code>
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>

### 18.2.12 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>

## 18.3 GPIO Firmware driver defines

### 18.3.1 GPIO

#### *GPIO Alternate function selection*

GPIO\_AF0\_RTC\_50Hz  
GPIO\_AF0\_MCO  
GPIO\_AF0\_TAMPER  
GPIO\_AF0\_SWJ  
GPIO\_AF0\_TRACE  
GPIO\_AF1\_TIM1  
GPIO\_AF1\_TIM2  
GPIO\_AF2\_TIM3  
GPIO\_AF2\_TIM4  
GPIO\_AF2\_TIM5  
GPIO\_AF3\_TIM8  
GPIO\_AF3\_TIM9  
GPIO\_AF3\_TIM10  
GPIO\_AF3\_TIM11  
GPIO\_AF4\_I2C1  
GPIO\_AF4\_I2C2  
GPIO\_AF4\_I2C3  
GPIO\_AF5\_SPI1  
GPIO\_AF5\_SPI2  
GPIO\_AF6\_SPI3  
GPIO\_AF7\_USART1  
GPIO\_AF7\_USART2  
GPIO\_AF7\_USART3  
GPIO\_AF8\_UART4  
GPIO\_AF8\_UART5  
GPIO\_AF8\_USART6  
GPIO\_AF9\_CAN1  
GPIO\_AF9\_CAN2

GPIO\_AF9\_TIM12  
GPIO\_AF9\_TIM13  
GPIO\_AF9\_TIM14  
GPIO\_AF10\_OTG\_FS  
GPIO\_AF10\_OTG\_HS  
GPIO\_AF11\_ETH  
GPIO\_AF12\_FSMC  
GPIO\_AF12\_OTG\_HS\_FS  
GPIO\_AF12\_SDIO  
GPIO\_AF13\_DCMI  
GPIO\_AF15\_EVENTOUT

**GPIO Exported Macros**

`_HAL_GPIO_EXTI_GET_FLAG`

**Description:**

- Checks whether the specified EXTI line flag is set or not.

**Parameters:**

- `_EXTI_LINE_`: specifies the EXTI line flag to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

**Return value:**

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

`_HAL_GPIO_EXTI_CLEAR_FLAG`

**Description:**

- Clears the EXTI's line pending flags.

**Parameters:**

- `_EXTI_LINE_`: specifies the EXTI lines flags to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

**Return value:**

- None

`_HAL_GPIO_EXTI_GET_IT`

**Description:**

- Checks whether the specified EXTI line is asserted or not.

**Parameters:**

- `_EXTI_LINE_`: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

**Return value:**

- The: new state of `_EXTI_LINE_` (SET or

RESET).

#### \_HAL\_GPIO\_EXTI\_CLEAR\_IT

**Description:**

- Clears the EXTI's line pending bits.

**Parameters:**

- \_EXTI\_LINE\_: specifies the EXTI lines to clear. This parameter can be any combination of GPIO\_PIN\_x where x can be (0..15)

**Return value:**

- None

#### \_HAL\_GPIO\_EXTI\_GENERATE\_SWIT

**Description:**

- Generates a Software interrupt on selected EXTI line.

**Parameters:**

- \_EXTI\_LINE\_: specifies the EXTI line to check. This parameter can be GPIO\_PIN\_x where x can be(0..15)

**Return value:**

- None

**GPIO mode define**

GPIO_MODE_INPUT	Input Floating Mode
GPIO_MODE_OUTPUT_PP	Output Push Pull Mode
GPIO_MODE_OUTPUT_OD	Output Open Drain Mode
GPIO_MODE_AF_PP	Alternate Function Push Pull Mode
GPIO_MODE_AF_OD	Alternate Function Open Drain Mode
GPIO_MODE_ANALOG	Analog Mode
GPIO_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
GPIO_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
GPIO_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
GPIO_MODE_EVT_RISING	External Event Mode with Rising edge trigger detection
GPIO_MODE_EVT_FALLING	External Event Mode with Falling edge trigger detection
GPIO_MODE_EVT_RISING_FALLING	External Event Mode with Rising/Falling edge trigger detection

**GPIO pins define**

#### GPIO\_PIN\_0

GPIO\_PIN\_1  
GPIO\_PIN\_2  
GPIO\_PIN\_3  
GPIO\_PIN\_4  
GPIO\_PIN\_5  
GPIO\_PIN\_6  
GPIO\_PIN\_7  
GPIO\_PIN\_8  
GPIO\_PIN\_9  
GPIO\_PIN\_10  
GPIO\_PIN\_11  
GPIO\_PIN\_12  
GPIO\_PIN\_13  
GPIO\_PIN\_14  
GPIO\_PIN\_15  
GPIO\_PIN\_All  
GPIO\_PIN\_MASK

***GPIO Private Constants***

GPIO\_MODE  
EXTI\_MODE  
GPIO\_MODE\_IT  
GPIO\_MODE\_EVT  
RISING\_EDGE  
FALLING\_EDGE  
GPIO\_OUTPUT\_TYPE  
GPIO\_NUMBER

***GPIO Private Macros***

IS\_GPIO\_PIN\_ACTION  
IS\_GPIO\_PIN  
IS\_GPIO\_MODE  
IS\_GPIO\_SPEED  
IS\_GPIO\_PULL

***GPIO pull define***

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

***GPIO speed define***

GPIO_SPEED_FREQ_LOW	IO works at 2 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_MEDIUM	range 12,5 MHz to 50 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ_HIGH	range 25 MHz to 100 MHz, please refer to the product datasheet
GPIO_SPEED_FREQ VERY HIGH	range 50 MHz to 200 MHz, please refer to the product datasheet

## 19 HAL GPIO Extension Driver

### 19.1 GPIOEx Firmware driver defines

#### 19.1.1 GPIOEx

*GPIO Get Port Index*

`GPIO_GET_INDEX`

*GPIO Check Alternate Function*

`IS_GPIO_AF`

## 20 HAL HASH Generic Driver

### 20.1 HASH Firmware driver registers structures

#### 20.1.1 HASH\_InitTypeDef

##### Data Fields

- *uint32\_t DataType*
- *uint32\_t KeySize*
- *uint8\_t \* pKey*

##### Field Documentation

- ***uint32\_t HASH\_InitTypeDef::DataType***  
32-bit data, 16-bit data, 8-bit data or 1-bit 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

#### 20.1.2 HASH\_HandleTypeDef

##### 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 input 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

## 20.2 HASH Firmware driver API description

### 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_MspInit()`:
  - a. Enable the HASH interface clock using `__HAL_RCC_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 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

This section contains the following APIs:

- [`HAL\_HASH\_MD5\_Start\(\)`](#)
- [`HAL\_HASH\_MD5\_Accumulate\(\)`](#)
- [`HAL\_HASH\_SHA1\_Start\(\)`](#)
- [`HAL\_HASH\_SHA1\_Accumulate\(\)`](#)

## 20.2.3 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

This section contains the following APIs:

- [`HAL\_HASH\_MD5\_Start\_IT\(\)`](#)
- [`HAL\_HASH\_SHA1\_Start\_IT\(\)`](#)
- [`HAL\_HASH\_IRQHandler\(\)`](#)
- [`HAL\_HMAC\_SHA1\_Start\(\)`](#)
- [`HAL\_HMAC\_MD5\_Start\(\)`](#)

## 20.2.4 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

This section contains the following APIs:

- [`HAL\_HASH\_MD5\_Start\_DMA\(\)`](#)
- [`HAL\_HASH\_MD5\_Finish\(\)`](#)
- [`HAL\_HASH\_SHA1\_Start\_DMA\(\)`](#)

- [\*HAL\\_HASH\\_SHA1\\_Finish\(\)\*](#)
- [\*HAL\\_HASH\\_SHA1\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_HASH\\_MD5\\_Start\\_IT\(\)\*](#)

### 20.2.5 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

This section contains the following APIs:

- [\*HAL\\_HMAC\\_MD5\\_Start\(\)\*](#)
- [\*HAL\\_HMAC\\_SHA1\\_Start\(\)\*](#)
- [\*HAL\\_HASH\\_SHA1\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HASH\\_SHA1\\_Finish\(\)\*](#)
- [\*HAL\\_HASH\\_MD5\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HASH\\_MD5\\_Finish\(\)\*](#)

### 20.2.6 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

This section contains the following APIs:

- [\*HAL\\_HMAC\\_MD5\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HMAC\\_SHA1\\_Start\\_DMA\(\)\*](#)

### 20.2.7 Peripheral State functions

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

This section contains the following APIs:

- [\*HAL\\_HASH\\_GetState\(\)\*](#)
- [\*HAL\\_HASH\\_IRQHandler\(\)\*](#)

### 20.2.8 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.

This section contains the following APIs:

- [\*HAL\\_HASH\\_Init\(\)\*](#)
- [\*HAL\\_HASH\\_DeInit\(\)\*](#)
- [\*HAL\\_HASH\\_MspInit\(\)\*](#)
- [\*HAL\\_HASH\\_MspDeInit\(\)\*](#)
- [\*HAL\\_HASH\\_InCpltCallback\(\)\*](#)
- [\*HAL\\_HASH\\_ErrorCallback\(\)\*](#)

- [\*HAL\\_HASH\\_DgstCpltCallback\(\)\*](#)

## 20.2.9 HAL\_HASH\_MD5\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Start (HASH_HandleTypeDef * 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>• HAL status</li> </ul>

## 20.2.10 HAL\_HASH\_MD5\_Accumulate

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t 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>• HAL status</li> </ul>

## 20.2.11 HAL\_HASH\_SHA1\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t 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> </ul>

- **Timeout:** Timeout value
- Return values HAL status

### 20.2.12 HAL\_HASH\_SHA1\_Accumulate

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</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> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Input buffer size in bytes must be a multiple of 4 otherwise the digest computation is corrupted.</li> </ul>

### 20.2.13 HAL\_HASH\_MD5\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)</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 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>• HAL status</li> </ul>

### 20.2.14 HAL\_HASH\_SHA1\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)</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> </ul>

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

### 20.2.15 HAL\_HASH\_IRQHandler

Function Name	<b>void HAL_HASH_IRQHandler (HASH_HandleTypeDef * 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>

### 20.2.16 HAL\_HMAC\_SHA1\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</b>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 20.2.17 HAL\_HMAC\_MD5\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_MD5_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</b>
Function Description	Initializes the HASH peripheral in HMAC 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 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>• HAL status</li> </ul>

### 20.2.18 HAL\_HASH\_MD5\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t 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>HAL status</li> </ul>

### 20.2.19 HAL\_HASH\_MD5\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t 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>HAL status</li> </ul>

### 20.2.20 HAL\_HASH\_SHA1\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t 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 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>HAL status</li> </ul>

### 20.2.21 HAL\_HASH\_SHA1\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t 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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

## 20.2.22 HAL\_HASH\_SHA1\_Start\_IT

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t 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>• HAL status</li> </ul>

## 20.2.23 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>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 16 bytes.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 20.2.24 HAL\_HMAC\_MD5\_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMAC_MD5_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in HMAC 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 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>• HAL status</li> </ul>

### 20.2.25 HAL\_HMAC\_SHA1\_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMAC_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 20.2.26 HAL\_HASH\_SHA1\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
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 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>• HAL status</li> </ul>

### 20.2.27 HAL\_HASH\_SHA1\_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)</code>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 20.2.28 HAL\_HASH\_MD5\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
---------------	--

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>HAL status</li> </ul>

### 20.2.29 HAL\_HASH\_MD5\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t 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>HAL status</li> </ul>

### 20.2.30 HAL\_HMAC\_MD5\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</b>
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 multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 20.2.31 HAL\_HMAC\_SHA1\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t 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>• HAL status</li> </ul>
---------------	--

### 20.2.32 HAL\_HASH\_GetState

Function Name	<b>HAL_HASH_STATETypeDef HAL_HASH_GetState (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>• HAL state</li> </ul>

### 20.2.33 HAL\_HASH\_IRQHandler

Function Name	<b>void HAL_HASH_IRQHandler (HASH_HandleTypeDef * 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>

### 20.2.34 HAL\_HASH\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HASH_Init (HASH_HandleTypeDef * hhash)</b>
Function Description	Initializes the HASH according to the specified parameters in the HASH_HandleTypeDef and creates the associated handle.
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>• HAL status</li> </ul>

### 20.2.35 HAL\_HASH\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_HASH_DelInit (HASH_HandleTypeDef * hhash)</b>
Function Description	Deinitializes the HASH peripheral.
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>• HAL status</li> </ul>

**Notes**

- This API must be called before starting a new processing.

### 20.2.36 HAL\_HASH\_MspInit

Function Name	<b>void HAL_HASH_MspInit (HASH_HandleTypeDef * hhash)</b>
Function Description	Initializes the 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 <b>20.2.37 HAL_HASH_MspDeInit</b> Function Name Function Description Parameters Return values	<ul style="list-style-type: none"> <li>• None</li> </ul> <p><b>void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)</b></p> <p>DeInitializes HASH MSP.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>20.2.38 HAL_HASH_InCpltCallback</b> Function Name Function Description Parameters Return values	<p><b>void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)</b></p> <p>Input data transfer complete callback.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>20.2.39 HAL_HASH_ErrorCallback</b> Function Name Function Description Parameters Return values	<p><b>void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)</b></p> <p>Data transfer Error callback.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>20.2.40 HAL_HASH_DgstCpltCallback</b> Function Name Function Description Parameters Return values Notes	<p><b>void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)</b></p> <p>Digest computation complete callback.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul> <ul style="list-style-type: none"> <li>• This callback is not relevant with DMA.</li> </ul>
<b>20.2.41 HAL_HASH_GetState</b> Function Name Function Description Parameters	<p><b>HAL_HASH_STATETTypeDef HAL_HASH_GetState (HASH_HandleTypeDef * hhash)</b></p> <p>return the HASH state</p> <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>• HAL state</li></ul>
---------------	---

#### 20.2.42 HAL\_HASH\_MspInit

Function Name      **void HAL\_HASH\_MspInit (HASH\_HandleTypeDef \* hhash)**

Function Description      Initializes the HASH MSP.

Parameters      

- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module

Return values      

- None

#### 20.2.43 HAL\_HASH\_MspDeInit

Function Name      **void HAL\_HASH\_MspDeInit (HASH\_HandleTypeDef \* hhash)**

Function Description      DeInitializes HASH MSP.

Parameters      

- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module

Return values      

- None

#### 20.2.44 HAL\_HASH\_InCpltCallback

Function Name      **void HAL\_HASH\_InCpltCallback (HASH\_HandleTypeDef \* hhash)**

Function Description      Input data transfer complete callback.

Parameters      

- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module

Return values      

- None

#### 20.2.45 HAL\_HASH\_DgstCpltCallback

Function Name      **void HAL\_HASH\_DgstCpltCallback (HASH\_HandleTypeDef \* hhash)**

Function Description      Digest computation complete callback.

Parameters      

- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module

Return values      

- None

Notes      

- This callback is not relevant with DMA.

#### 20.2.46 HAL\_HASH\_ErrorCallback

Function Name      **void HAL\_HASH\_ErrorCallback (HASH\_HandleTypeDef \* hhash)**

Function Description      Data transfer Error callback.

Parameters      

- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module

Return values      

- None

## 20.3 HASH Firmware driver defines

### 20.3.1 HASH

#### ***HASH Data Type***

**HASH\_DATATYPE\_32B** 32-bit data. No swapping  
**HASH\_DATATYPE\_16B** 16-bit data. Each half word is swapped  
**HASH\_DATATYPE\_8B** 8-bit data. All bytes are swapped  
**HASH\_DATATYPE\_1B** 1-bit data. In the word all bits are swapped

#### ***HASH Algorithm Selection***

**HASH\_ALGOSELECTION\_SHA1** HASH function is SHA1

**HASH\_ALGOSELECTION\_MD5** HASH function is MD5

#### ***HASH Algorithm Mode***

**HASH\_ALGOMODE\_HASH** Algorithm is HASH

**HASH\_ALGOMODE\_HMAC** Algorithm is HMAC

#### ***HASH HMAC Long key***

**HASH\_HMAC\_KEYTYPE\_SHORTKEY** HMAC Key is <= 64 bytes

**HASH\_HMAC\_KEYTYPE\_LONGKEY** HMAC Key is > 64 bytes

#### ***HASH Flags definition***

**HASH\_FLAG\_DINIS** 16 locations are free in the DIN : A new block can be entered into the input buffer  
**HASH\_FLAG\_DCIS** Digest calculation complete  
**HASH\_FLAG\_DMAS** DMA interface is enabled (DMAE=1) or a transfer is ongoing  
**HASH\_FLAG\_BUSY** The hash core is Busy : processing a block of data  
**HASH\_FLAG\_DINNE** DIN not empty : The input buffer contains at least one word of data

#### ***HASH Interrupts definition***

**HASH\_IT\_DINI** A new block can be entered into the input buffer (DIN)  
**HASH\_IT\_DCI** Digest calculation complete

#### ***HASH Exported Macros***

##### **\_HAL\_HASH\_RESET\_HANDLE\_STATE** **Description:**

- Reset HASH handle state.

##### **Parameters:**

- **\_HANDLE\_**: specifies the HASH handle.

##### **Return value:**

- None

##### **\_HAL\_HASH\_GET\_FLAG**

##### **Description:**

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

**Parameters:**

- `__FLAG__`: specifies the flag to check.  
This parameter can be one of the following values:
  - HASH\_FLAG\_DINIS: A new block can be entered into the input buffer.
  - HASH\_FLAG\_DCIS: Digest calculation complete
  - HASH\_FLAG\_DMAS: DMA interface is enabled (DMAE=1) or a transfer is ongoing
  - HASH\_FLAG\_BUSY: The hash core is Busy : processing a block of data
  - HASH\_FLAG\_DINNE: DIN not empty : The input buffer contains at least one word of data

**Return value:**

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

`_HAL_HASH_START_DIGEST`

**Description:**

- Start the digest computation.

**Return value:**

- None

`_HAL_HASH_SET_NBVALIDBITS`

**Description:**

- Set the number of valid bits in last word written in Data register.

**Parameters:**

- SIZE: size in byte of last data written in Data register.

**Return value:**

- None

***HASH Private Macros***

`IS_HASH_ALGOSELECTION`  
`IS_HASH_ALGOMODE`  
`IS_HASH_DATATYPE`  
`IS_HASH_HMAC_KEYTYPE`  
`IS_HASH_SHA1_BUFFER_SIZE`

## 21 HAL HCD Generic Driver

### 21.1 HCD Firmware driver registers structures

#### 21.1.1 HCD\_HandleTypeDef

##### Data Fields

- *HCD\_TypeDef \* Instance*
- *HCD\_InitTypeDef Init*
- *HCD\_HCTypedef hc*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HCD\_StateTypeDef State*
- *void \* pData*

##### Field Documentation

- ***HCD\_TypeDef\* HCD\_HandleTypeDef::Instance***  
Register base address
- ***HCD\_InitTypeDef HCD\_HandleTypeDef::Init***  
HCD required parameters
- ***HCD\_HCTypedef HCD\_HandleTypeDef::hc[15]***  
Host channels parameters
- ***HAL\_LockTypeDef HCD\_HandleTypeDef::Lock***  
HCD peripheral status
- ***\_\_IO HCD\_StateTypeDef HCD\_HandleTypeDef::State***  
HCD communication state
- ***void\* HCD\_HandleTypeDef::pData***  
Pointer Stack Handler

### 21.2 HCD Firmware driver API description

#### 21.2.1 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\_MsplInit() API:
  - a. Enable the HCD/USB Low Level interface clock using the following macros
    - `__HAL_RCC_USB_OTG_FS_CLK_ENABLE();`
    - `__HAL_RCC_USB_OTG_HS_CLK_ENABLE();` (For High Speed Mode)
    - `__HAL_RCC_USB_OTG_HS_ULPI_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();

### 21.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- [\*HAL\\_HCD\\_Init\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_Init\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_Halt\(\)\*](#)
- [\*HAL\\_HCD\\_DelInit\(\)\*](#)
- [\*HAL\\_HCD\\_MspInit\(\)\*](#)
- [\*HAL\\_HCD\\_MspDelInit\(\)\*](#)

### 21.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the USB Host Data Transfer

This section contains the following APIs:

- [\*HAL\\_HCD\\_HC\\_SubmitRequest\(\)\*](#)
- [\*HAL\\_HCD\\_IRQHandler\(\)\*](#)
- [\*HAL\\_HCD\\_SOF\\_Callback\(\)\*](#)
- [\*HAL\\_HCD\\_Connect\\_Callback\(\)\*](#)
- [\*HAL\\_HCD\\_Disconnect\\_Callback\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_NotifyURBChange\\_Callback\(\)\*](#)

### 21.2.4 Peripheral Control functions

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

This section contains the following APIs:

- [\*HAL\\_HCD\\_Start\(\)\*](#)
- [\*HAL\\_HCD\\_Stop\(\)\*](#)
- [\*HAL\\_HCD\\_ResetPort\(\)\*](#)

### 21.2.5 Peripheral State functions

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

This section contains the following APIs:

- [\*HAL\\_HCD\\_GetState\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_GetURBState\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_GetXferCount\(\)\*](#)
- [\*HAL\\_HCD\\_HC\\_GetState\(\)\*](#)
- [\*HAL\\_HCD\\_GetCurrentFrame\(\)\*](#)
- [\*HAL\\_HCD\\_GetCurrentSpeed\(\)\*](#)

### 21.2.6 HAL\_HCD\_Init

Function Name                    **HAL\_StatusTypeDef HAL\_HCD\_Init (HCD\_HandleTypeDef \* hhcd)**

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>• HAL status</li> </ul>
<b>21.2.7 HAL_HCD_HC_Init</b>	
Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDefDef * 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 to32K</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>21.2.8 HAL_HCD_HC_Halt</b>	
Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDefDef * 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>• HAL status</li> </ul>
<b>21.2.9 HAL_HCD_DelInit</b>	
Function Name	<b>HAL_StatusTypeDef HAL_HCD_DelInit (HCD_HandleTypeDefDef * hhcd)</b>
Function Description	Deinitialize 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>• HAL status</li> </ul>
<b>21.2.10 HAL_HCD_MspInit</b>	

Function Name	<b>void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)</b>
Function Description	Initialize 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>

### 21.2.11 HAL\_HCD\_MspDelInit

Function Name	<b>void HAL_HCD_MspDelInit (HCD_HandleTypeDef * hhcd)</b>
Function Description	Deinitialize 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>

### 21.2.12 HAL\_HCD\_HC\_SubmitRequest

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)</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>• HAL status</li> </ul>

### 21.2.13 HAL\_HCD\_IRQHandler

Function Name	<b>void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)</b>
Function Description	Handle 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>

### 21.2.14 HAL\_HCD\_SOF\_Callback

Function Name	<b>void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * 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>
<b>21.2.15 HAL_HCD_Connect_Callback</b>		
	Function Name	<b>void HAL_HCD_Connect_Callback (HCD_HandleTypeDefDef * hhcd)</b>
	Function Description	Connection 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>
<b>21.2.16 HAL_HCD_Disconnect_Callback</b>		
	Function Name	<b>void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDefDef * hhcd)</b>
	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>
<b>21.2.17 HAL_HCD_HC_NotifyURBChange_Callback</b>		
	Function Name	<b>void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDefDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)</b>
	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>
<b>21.2.18 HAL_HCD_Start</b>		
	Function Name	<b>HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDefDef * hhcd)</b>
	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>• HAL status</li> </ul>
<b>21.2.19 HAL_HCD_Stop</b>		
	Function Name	<b>HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDefDef * 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>• HAL status</li> </ul>

### 21.2.20 HAL\_HCD\_ResetPort

Function Name	<b>HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * 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>• HAL status</li> </ul>

### 21.2.21 HAL\_HCD\_GetState

Function Name	<b>HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef * hhcd)</b>
Function Description	Return the HCD handle 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>• HAL state</li> </ul>

### 21.2.22 HAL\_HCD\_HC\_GetURBState

Function Name	<b>HCD_URBStateTypeDef HAL_HCD_HC_GetURBState (HCD_HandleTypeDef * 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>• URB state. This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL</li> </ul>

### 21.2.23 HAL\_HCD\_HC\_GetXferCount

Function Name	<b>uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef * 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>• last transfer size in byte</li> </ul>

### 21.2.24 HAL\_HCD\_HC\_GetState

Function Name	<b>HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef * hhcd, uint8_t chnum)</b>
---------------	---

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>• Host channel state This parameter can be one of these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR</li> </ul>

### 21.2.25 HAL\_HCD\_GetCurrentFrame

Function Name	<code>uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)</code>
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>• Current Host frame number</li> </ul>

### 21.2.26 HAL\_HCD\_GetCurrentSpeed

Function Name	<code>uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)</code>
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>• Enumeration speed</li> </ul>

## 21.3 HCD Firmware driver defines

### 21.3.1 HCD

#### *HCD Exported Macros*

`_HAL_HCD_ENABLE`  
`_HAL_HCD_DISABLE`  
`_HAL_HCD_GET_FLAG`  
`_HAL_HCD_CLEAR_FLAG`  
`_HAL_HCD_IS_INVALID_INTERRUPT`  
`_HAL_HCD_CLEAR_HC_INT`  
`_HAL_HCD_MASK_HALT_HC_INT`  
`_HAL_HCD_UNMASK_HALT_HC_INT`  
`_HAL_HCD_MASK_ACK_HC_INT`  
`_HAL_HCD_UNMASK_ACK_HC_INT`

#### *HCD Instance definition*

`IS_HCD_ALL_INSTANCE`

#### *HCD PHY Module*

HCD\_PHY\_ULPI  
HCD\_PHY\_EMBEDDED  
*HCD Speed*  
HCD\_SPEED\_HIGH  
HCD\_SPEED\_LOW  
HCD\_SPEED\_FULL

## 22 HAL I2C Generic Driver

### 22.1 I2C Firmware driver registers structures

#### 22.1.1 I2C\_InitTypeDef

##### 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\*](#)

#### 22.1.2 I2C\_HandleTypeDef

##### Data Fields

- *I2C\_TypeDef \* Instance*
- *I2C\_InitTypeDef Init*
- *uint8\_t \* pBuffPtr*
- *uint16\_t XferSize*
- *\_IO uint16\_t XferCount*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *HAL\_LockTypeDef Lock*
- *\_IO HAL\_I2C\_StateTypeDef State*
- *\_IO uint32\_t ErrorCode*

#### Field Documentation

- *I2C\_TypeDef\* I2C\_HandleTypeDef::Instance*  
I2C registers base address
- *I2C\_InitTypeDef I2C\_HandleTypeDef::Init*  
I2C communication parameters
- *uint8\_t\* I2C\_HandleTypeDef::pBuffPtr*  
Pointer to I2C transfer buffer
- *uint16\_t I2C\_HandleTypeDef::XferSize*  
I2C transfer size
- *\_IO uint16\_t I2C\_HandleTypeDef::XferCount*  
I2C transfer counter
- *DMA\_HandleTypeDef\* I2C\_HandleTypeDef::hdmatx*  
I2C Tx DMA handle parameters
- *DMA\_HandleTypeDef\* I2C\_HandleTypeDef::hdmarx*  
I2C Rx DMA handle parameters
- *HAL\_LockTypeDef I2C\_HandleTypeDef::Lock*  
I2C locking object
- *\_IO HAL\_I2C\_StateTypeDef I2C\_HandleTypeDef::State*  
I2C communication state
- *\_IO uint32\_t I2C\_HandleTypeDef::ErrorCode*  
I2C Error code

## 22.2 I2C Firmware driver API description

### 22.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C\_HandleTypeDef handle structure, for example: I2C\_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implement the HAL\_I2C\_MspInit() API:
  - 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

#### 22.2.2 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\_DelInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- [\*\*HAL\\_I2C\\_Init\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_MspDelInit\(\)\*\*](#)

#### 22.2.3 IO operation functions

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

1. There are two modes of transfer:

- Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
- HAL\_I2C\_Master\_Transmit()
  - HAL\_I2C\_Master\_Receive()
  - HAL\_I2C\_Slave\_Transmit()
  - HAL\_I2C\_Slave\_Receive()
  - HAL\_I2C\_Mem\_Write()
  - HAL\_I2C\_Mem\_Read()
  - HAL\_I2C\_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
- HAL\_I2C\_Master\_Transmit\_IT()
  - HAL\_I2C\_Master\_Receive\_IT()
  - HAL\_I2C\_Slave\_Transmit\_IT()
  - HAL\_I2C\_Slave\_Receive\_IT()
  - HAL\_I2C\_Mem\_Write\_IT()
  - HAL\_I2C\_Mem\_Read\_IT()
4. No-Blocking mode functions with DMA are :
- HAL\_I2C\_Master\_Transmit\_DMA()
  - HAL\_I2C\_Master\_Receive\_DMA()
  - HAL\_I2C\_Slave\_Transmit\_DMA()
  - HAL\_I2C\_Slave\_Receive\_DMA()
  - HAL\_I2C\_Mem\_Write\_DMA()
  - HAL\_I2C\_Mem\_Read\_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
- HAL\_I2C\_MemTxCpltCallback()
  - HAL\_I2C\_MemRxCpltCallback()
  - HAL\_I2C\_MasterTxCpltCallback()
  - HAL\_I2C\_MasterRxCpltCallback()
  - HAL\_I2C\_SlaveTxCpltCallback()
  - HAL\_I2C\_SlaveRxCpltCallback()
  - HAL\_I2C\_ErrorCallback()

This section contains the following APIs:

- [\*\*HAL\\_I2C\\_Master\\_Transmit\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Receive\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Transmit\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Receive\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Transmit\\_IT\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Receive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Transmit\\_IT\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Receive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Transmit\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Master\\_Receive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Transmit\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Slave\\_Receive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Mem\\_Write\(\)\*\*](#)
- [\*\*HAL\\_I2C\\_Mem\\_Read\(\)\*\*](#)

- `HAL_I2C_Mem_Write_IT()`
- `HAL_I2C_Mem_Read_IT()`
- `HAL_I2C_Mem_Write_DMA()`
- `HAL_I2C_Mem_Read_DMA()`
- `HAL_I2C_IsDeviceReady()`
- `HAL_I2C_EV_IRQHandler()`
- `HAL_I2C_ER_IRQHandler()`
- `HAL_I2C_MasterTxCpltCallback()`
- `HAL_I2C_MasterRxCpltCallback()`
- `HAL_I2C_SlaveTxCpltCallback()`
- `HAL_I2C_SlaveRxCpltCallback()`
- `HAL_I2C_MemTxCpltCallback()`
- `HAL_I2C_MemRxCpltCallback()`
- `HAL_I2C_ErrorCallback()`

## 22.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- `HAL_I2C_GetState()`
- `HAL_I2C_GetError()`

## 22.2.5 HAL\_I2C\_Init

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)</code>
Function Description	Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <code>hi2c</code>: 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>• HAL status</li> </ul>

## 22.2.6 HAL\_I2C\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)</code>
Function Description	DeInitializes the I2C peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <code>hi2c</code>: 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>• HAL status</li> </ul>

## 22.2.7 HAL\_I2C\_MspInit

Function Name	<code>void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)</code>
Function Description	I2C MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <code>hi2c</code>: 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>

## 22.2.8 HAL\_I2C\_MspDeInit



Function Name	<b>void HAL_I2C_MspDelInit (I2C_HandleTypeDef * 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>

### 22.2.9 HAL\_I2C\_Master\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * 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>• HAL status</li> </ul>

### 22.2.10 HAL\_I2C\_Master\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive (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>• HAL status</li> </ul>

### 22.2.11 HAL\_I2C\_Slave\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit (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> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 22.2.12 HAL\_I2C\_Slave\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 22.2.13 HAL\_I2C\_Master\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 22.2.14 HAL\_I2C\_Master\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</b>
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>• HAL status</li> </ul>

## 22.2.15 HAL\_I2C\_Slave\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (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 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> </ul>

- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- HAL status

### 22.2.16 HAL\_I2C\_Slave\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (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 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	• HAL status

### 22.2.17 HAL\_I2C\_Master\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</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	• HAL status

### 22.2.18 HAL\_I2C\_Master\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)</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> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> </ul>
Return values	• HAL status

### 22.2.19 HAL\_I2C\_Slave\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (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>HAL status</li> </ul>

## 22.2.20 HAL\_I2C\_Slave\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA(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>HAL status</li> </ul>

## 22.2.21 HAL\_I2C\_Mem\_Write

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

## 22.2.22 HAL\_I2C\_Mem\_Read

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
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> </ul>

	<ul style="list-style-type: none"> <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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 22.2.23 HAL\_I2C\_Mem\_Write\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</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 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>HAL status</li> </ul>

### 22.2.24 HAL\_I2C\_Mem\_Read\_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</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 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>HAL status</li> </ul>

### 22.2.25 HAL\_I2C\_Mem\_Write\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</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>HAL status</li> </ul>

## 22.2.26 HAL\_I2C\_Mem\_Read\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</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>HAL status</li> </ul>

## 22.2.27 HAL\_I2C\_IsDeviceReady

Function Name	<code>HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t 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>HAL status</li> </ul>
Notes	This function is used with Memory devices

## 22.2.28 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>HAL status</li> </ul>

**22.2.29 HAL\_I2C\_ER\_IRQHandler**

Function Name	<b>void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)</b>
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>• HAL status</li></ul>

**22.2.30 HAL\_I2C\_MasterTxCpltCallback**

Function Name	<b>void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)</b>
Function Description	Master Tx 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>

**22.2.31 HAL\_I2C\_MasterRxCpltCallback**

Function Name	<b>void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)</b>
Function Description	Master 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>

**22.2.32 HAL\_I2C\_SlaveTxCpltCallback**

Function Name	<b>void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)</b>
Function Description	Slave Tx 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>

**22.2.33 HAL\_I2C\_SlaveRxCpltCallback**

Function Name	<b>void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)</b>
Function Description	Slave 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>

**22.2.34 HAL\_I2C\_MemTxCpltCallback**

---

Function Name	<b>void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)</b>
Function Description	Memory Tx 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>

### 22.2.35 HAL\_I2C\_MemRxCpltCallback

Function Name	<b>void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)</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>

### 22.2.36 HAL\_I2C\_ErrorCallback

Function Name	<b>void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * 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>

### 22.2.37 HAL\_I2C\_GetState

Function Name	<b>HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * 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>• HAL state</li> </ul>

### 22.2.38 HAL\_I2C\_GetError

Function Name	<b>uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)</b>
Function Description	Return the I2C error code.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c:</b> : 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>• I2C Error Code</li> </ul>

## 22.3 I2C Firmware driver defines

### 22.3.1 I2C

*I2C addressing mode*

`I2C_ADDRESSINGMODE_7BIT`  
`I2C_ADDRESSINGMODE_10BIT`

***I2C dual addressing mode***

`I2C_DUALADDRESS_DISABLE`  
`I2C_DUALADDRESS_ENABLE`

***I2C duty cycle in fast mode***

`I2C_DUTYCYCLE_2`  
`I2C_DUTYCYCLE_16_9`

***I2C Error Code***

<code>HAL_I2C_ERROR_NONE</code>	No error
<code>HAL_I2C_ERROR_BERR</code>	BERR error
<code>HAL_I2C_ERROR_ARLO</code>	ARLO error
<code>HAL_I2C_ERROR_AF</code>	AF error
<code>HAL_I2C_ERROR_OVR</code>	OVR error
<code>HAL_I2C_ERROR_DMA</code>	DMA transfer error
<code>HAL_I2C_ERROR_TIMEOUT</code>	Timeout Error

***I2C Exported Macros***

`__HAL_I2C_RESET_HANDLE_STATE` **Description:**

- Reset I2C handle state.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.

**Return value:**

- None

`__HAL_I2C_ENABLE_IT` **Description:**

- Enable or disable the specified I2C interrupts.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.
- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - `I2C_IT_BUF`: Buffer interrupt enable
  - `I2C_IT_EVT`: Event interrupt enable
  - `I2C_IT_ERR`: Error interrupt enable

**Return value:**

- None

---

`__HAL_I2C_DISABLE_IT`  
`__HAL_I2C_GET_IT_SOURCE`

**Description:**

- Checks if the specified I2C interrupt source is enabled or disabled.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle. This parameter can be I2Cx where x: 1, 2, or 3 to select the I2C peripheral.
- `__INTERRUPT__`: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - `I2C_IT_BUF`: Buffer interrupt enable
  - `I2C_IT_EVT`: Event interrupt enable
  - `I2C_IT_ERR`: Error interrupt enable

**Return value:**

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

`__HAL_I2C_GET_FLAG`

**Description:**

- Checks whether the specified I2C flag is set or not.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle. This parameter can be I2Cx where x: 1, 2, or 3 to select the I2C peripheral.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `I2C_FLAG_SMBALERT`: SMBus Alert flag
  - `I2C_FLAG_TIMEOUT`: Timeout or Tlow error flag
  - `I2C_FLAG_PECERR`: PEC error in reception flag
  - `I2C_FLAG_OVR`: Overrun/Underrun flag
  - `I2C_FLAG_AF`: Acknowledge failure flag
  - `I2C_FLAG_ARLO`: Arbitration lost flag
  - `I2C_FLAG_BERR`: Bus error flag
  - `I2C_FLAG_TXE`: Data register empty flag
  - `I2C_FLAG_RXNE`: Data register not empty flag
  - `I2C_FLAG_STOPF`: Stop detection flag
  - `I2C_FLAG_ADD10`: 10-bit header sent flag
  - `I2C_FLAG_BTF`: Byte transfer finished flag
  - `I2C_FLAG_ADDR`: Address sent flag

- Address matched flag
- I2C\_FLAG\_SB: Start bit flag
- I2C\_FLAG\_DUALF: Dual flag
- I2C\_FLAG\_SMBHOST: SMBus host header
- I2C\_FLAG\_SMBDEFAULT: SMBus default header
- I2C\_FLAG\_GENCALL: General call header flag
- I2C\_FLAG\_TRA: Transmitter/Receiver flag
- I2C\_FLAG\_BUSY: Bus busy flag
- I2C\_FLAG\_MSL: Master/Slave flag

**Return value:**

- The new state of \_\_FLAG\_\_ (TRUE or FALSE).

**\_HAL\_I2C\_CLEAR\_FLAG****Description:**

- Clears the I2C pending flags which are cleared by writing 0 in a specific bit.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.
- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C\_FLAG\_SMBALERT: SMBus Alert flag
  - I2C\_FLAG\_TIMEOUT: Timeout or Tlow error flag
  - I2C\_FLAG\_PECERR: PEC error in reception flag
  - I2C\_FLAG\_OVR: Overrun/Underrun flag (Slave mode)
  - I2C\_FLAG\_AF: Acknowledge failure flag
  - I2C\_FLAG\_ARLO: Arbitration lost flag (Master mode)
  - I2C\_FLAG\_BERR: Bus error flag

**Return value:**

- None

**\_HAL\_I2C\_CLEAR\_ADDRFLAG****Description:**

- Clears the I2C ADDR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle. This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.

**Return value:**

- None

`__HAL_I2C_CLEAR_STOPFLAG`

- Clears the I2C STOPF pending flag.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.  
This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.

**Return value:**

- None

`__HAL_I2C_ENABLE`

`__HAL_I2C_DISABLE`

***I2C Flag definition***

`I2C_FLAG_SMBALERT`

`I2C_FLAG_TIMEOUT`

`I2C_FLAG_PECERR`

`I2C_FLAG_OVR`

`I2C_FLAG_AF`

`I2C_FLAG_ARLO`

`I2C_FLAG_BERR`

`I2C_FLAG_TXE`

`I2C_FLAG_RXNE`

`I2C_FLAG_STOPF`

`I2C_FLAG_ADD10`

`I2C_FLAG_BTF`

`I2C_FLAG_ADDR`

`I2C_FLAG_SB`

`I2C_FLAG_DUALF`

`I2C_FLAG_SMBHOST`

`I2C_FLAG_SMBDEFAULT`

`I2C_FLAG_GENCALL`

`I2C_FLAG_TRA`

`I2C_FLAG_BUSY`

`I2C_FLAG_MSL`

***I2C general call addressing mode***

`I2C_GENERALCALL_DISABLE`

`I2C_GENERALCALL_ENABLE`

***I2C Interrupt configuration definition***

I2C\_IT\_BUF

I2C\_IT\_EVT

I2C\_IT\_ERR

***I2C Private macros to check input parameters***

IS\_I2C\_DUTY\_CYCLE

IS\_I2C\_ADDRESSING\_MODE

IS\_I2C\_DUAL\_ADDRESS

IS\_I2C\_GENERAL\_CALL

IS\_I2C\_NO\_STRETCH

IS\_I2C\_MEMADD\_SIZE

IS\_I2C\_CLOCK\_SPEED

IS\_I2C\_OWN\_ADDRESS1

IS\_I2C\_OWN\_ADDRESS2

***I2C Memory Address Size***

I2C\_MEMADD\_SIZE\_8BIT

I2C\_MEMADD\_SIZE\_16BIT

***I2C nostretch mode***

I2C\_NOSTRETCH\_DISABLE

I2C\_NOSTRETCH\_ENABLE

***I2C Private Constants***

I2C\_TIMEOUT\_FLAG

I2C\_TIMEOUT\_ADDR\_SLAVE

I2C\_TIMEOUT\_BUSY\_FLAG

I2C\_FLAG\_MASK

***I2C Private Macros***

I2C\_FREQRANGE

I2C\_RISE\_TIME

I2C\_SPEED\_STANDARD

I2C\_SPEED\_FAST

I2C\_SPEED

I2C\_7BIT\_ADD\_WRITE

I2C\_7BIT\_ADD\_READ

I2C\_10BIT\_ADDRESS

I2C\_10BIT\_HEADER\_WRITE

I2C\_10BIT\_HEADER\_READ

I2C\_MEM\_ADD\_MSB

I2C\_MEM\_ADD\_LSB



## 23 HAL I2S Generic Driver

### 23.1 I2S Firmware driver registers structures

#### 23.1.1 I2S\_InitTypeDef

##### Data Fields

- *uint32\_t Mode*
- *uint32\_t Standard*
- *uint32\_t DataFormat*
- *uint32\_t MCLKOutput*
- *uint32\_t AudioFreq*
- *uint32\_t CPOL*
- *uint32\_t ClockSource*

##### 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](#)

#### 23.1.2 I2S\_HandleTypeDef

##### Data Fields

- *SPI\_TypeDef \* Instance*
- *I2S\_InitTypeDef Init*
- *uint16\_t \* pTxBuffPtr*
- *\_\_IO uint16\_t TxXferSize*
- *\_\_IO uint16\_t TxXferCount*
- *uint16\_t \* pRxBuffPtr*

- `__IO uint16_t RxXferSize`
- `__IO uint16_t RxXferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `__IO HAL_LockTypeDef Lock`
- `__IO HAL_I2S_StateTypeDef State`
- `__IO uint32_t ErrorCode`

#### Field Documentation

- `SPI_HandleTypeDef* I2S_HandleTypeDef::Instance`
- `I2S_InitTypeDef I2S_HandleTypeDef::Init`
- `uint16_t* I2S_HandleTypeDef::pTxBuffPtr`
- `__IO uint16_t I2S_HandleTypeDef::TxXferSize`
- `__IO uint16_t I2S_HandleTypeDef::TxXferCount`
- `uint16_t* I2S_HandleTypeDef::pRxBuffPtr`
- `__IO uint16_t I2S_HandleTypeDef::RxXferSize`
- `__IO uint16_t I2S_HandleTypeDef::RxXferCount`
- `DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx`
- `__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock`
- `__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State`
- `__IO uint32_t I2S_HandleTypeDef::ErrorCode`

## 23.2 I2S Firmware driver API description

### 23.2.1 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 stm32f2xx\_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

### 23.2.2 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
- Call the function `HAL_I2S_DelInit()` to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- `HAL\_I2S\_Init\(\)`
- `HAL\_I2S\_DelInit\(\)`
- `HAL\_I2S\_MspInit\(\)`
- `HAL\_I2S\_MspDelInit\(\)`

### 23.2.3 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()

This section contains the following APIs:

- [\*HAL\\_I2S\\_Transmit\(\)\*](#)
- [\*HAL\\_I2S\\_Receive\(\)\*](#)
- [\*HAL\\_I2S\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_I2S\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_I2S\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_I2S\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_I2S\\_DMAPause\(\)\*](#)
- [\*HAL\\_I2S\\_DMAResume\(\)\*](#)
- [\*HAL\\_I2S\\_DMAStop\(\)\*](#)
- [\*HAL\\_I2S\\_IRQHandler\(\)\*](#)
- [\*HAL\\_I2S\\_TxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_I2S\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_I2S\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_I2S\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_I2S\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_I2S\\_GetState\(\)\*](#)
- [\*HAL\\_I2S\\_GetError\(\)\*](#)

### 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.

This section contains the following APIs:

- [\*HAL\\_I2S\\_GetState\(\)\*](#)
- [\*HAL\\_I2S\\_GetError\(\)\*](#)

### 23.2.5 HAL\_I2S\_Init

Function Name	<a href="#"><b>HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)</b></a>
Function Description	Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.
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>• HAL status</li> </ul>

### 23.2.6 HAL\_I2S\_DeInit

Function Name	<a href="#"><b>HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)</b></a>
Function Description	DeInitializes the I2S peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2s:</b> pointer to a I2S_HandleTypeDef structure that contains</li> </ul>

the configuration information for I2S module

Return values • HAL status

### 23.2.7 HAL\_I2S\_MspInit

Function Name **void HAL\_I2S\_MspInit (I2S\_HandleTypeDef \* hi2s)**

Function Description I2S MSP Init.

Parameters • **hi2s:** pointer to a I2S\_HandleTypeDef structure that contains the configuration information for I2S module

Return values • None

### 23.2.8 HAL\_I2S\_MspDeInit

Function Name **void HAL\_I2S\_MspDeInit (I2S\_HandleTypeDef \* hi2s)**

Function Description I2S MSP DeInit.

Parameters • **hi2s:** pointer to a I2S\_HandleTypeDef structure that contains the configuration information for I2S module

Return values • None

### 23.2.9 HAL\_I2S\_Transmit

Function Name **HAL\_StatusTypeDef HAL\_I2S\_Transmit (I2S\_HandleTypeDef \* hi2s, uint16\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

Function Description Transmit an amount of data in blocking mode.

Parameters • **hi2s:** pointer to a I2S\_HandleTypeDef structure that contains the configuration information for I2S module  
• **pData:** a 16-bit pointer to data buffer.  
• **Size:** number of data sample to be sent:  
• **Timeout:** Timeout duration

Return values • HAL status

Notes • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the **Size** parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the **Size** parameter means the number of 16-bit data length.  
• The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

### 23.2.10 HAL\_I2S\_Receive

Function Name **HAL\_StatusTypeDef HAL\_I2S\_Receive (I2S\_HandleTypeDef \* hi2s, uint16\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

Function Description Receive an amount of data in blocking mode.

Parameters • **hi2s:** pointer to a I2S\_HandleTypeDef structure that contains the configuration information for I2S module

	<ul style="list-style-type: none"> <li><b>pData:</b> a 16-bit pointer to data buffer.</li> <li><b>Size:</b> number of data sample to be sent:</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</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> <li>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.</li> </ul>

### 23.2.11 HAL\_I2S\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_IT (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	<ul style="list-style-type: none"> <li>HAL status</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>

### 23.2.12 HAL\_I2S\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_IT (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> <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>HAL status</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</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).
- 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.

### 23.2.13 HAL\_I2S\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_DMA (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>• HAL status</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>

### 23.2.14 HAL\_I2S\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_DMA (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>• HAL status</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>

### 23.2.15 HAL\_I2S\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DM_PAUSE (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>• HAL status</li> </ul>

### 23.2.16 HAL\_I2S\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAResume (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>• HAL status</li> </ul>

### 23.2.17 HAL\_I2S\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAStop (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>• HAL status</li> </ul>

### 23.2.18 HAL\_I2S\_IRQHandler

Function Name	<b>void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)</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>

### 23.2.19 HAL\_I2S\_TxHalfCpltCallback

Function Name	<b>void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)</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 contains the configuration information for I2S module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 23.2.20 HAL\_I2S\_TxCpltCallback

Function Name	<b>void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)</b>
---------------	---

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>

### 23.2.21 HAL\_I2S\_RxHalfCpltCallback

Function Name	<b>void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)</b>
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>

### 23.2.22 HAL\_I2S\_RxCpltCallback

Function Name	<b>void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)</b>
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>

### 23.2.23 HAL\_I2S\_ErrorCallback

Function Name	<b>void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * 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>

### 23.2.24 HAL\_I2S\_GetState

Function Name	<b>HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * 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>• HAL state</li> </ul>

### 23.2.25 HAL\_I2S\_GetError

Function Name	<b>uint32_t HAL_I2S_GetError (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>• I2S Error Code</li> </ul>

### 23.2.26 HAL\_I2S\_GetState

Function Name	<b>HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * 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>• HAL state</li></ul>

### 23.2.27 HAL\_I2S\_GetError

Function Name	<b>uint32_t HAL_I2S_GetError (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>• I2S Error Code</li></ul>

## 23.3 I2S Firmware driver defines

### 23.3.1 I2S

#### *I2S Audio Frequency*

I2S\_AUDIOFREQ\_192K  
I2S\_AUDIOFREQ\_96K  
I2S\_AUDIOFREQ\_48K  
I2S\_AUDIOFREQ\_44K  
I2S\_AUDIOFREQ\_32K  
I2S\_AUDIOFREQ\_22K  
I2S\_AUDIOFREQ\_16K  
I2S\_AUDIOFREQ\_11K  
I2S\_AUDIOFREQ\_8K  
I2S\_AUDIOFREQ\_DEFAULT

#### *I2S Clock Polarity*

I2S\_CPOL\_LOW  
I2S\_CPOL\_HIGH

#### *I2S Clock Source*

I2S\_CLOCK\_PLL  
I2S\_CLOCK\_EXTERNAL

#### *I2S Data Format*

I2S\_DATAFORMAT\_16B  
I2S\_DATAFORMAT\_16B\_EXTENDED

`I2S_DATAFORMAT_24B``I2S_DATAFORMAT_32B`***I2S Error Code***

<code>HAL_I2S_ERROR_NONE</code>	No error
<code>HAL_I2S_ERROR_UDR</code>	I2S Underrun error
<code>HAL_I2S_ERROR_OVR</code>	I2S Overrun error
<code>HAL_I2SEX_ERROR_UDR</code>	I2S extended Underrun error
<code>HAL_I2SEX_ERROR_OVR</code>	I2S extended Overrun error
<code>HAL_I2S_ERROR_FRE</code>	I2S Frame format error
<code>HAL_I2S_ERROR_DMA</code>	DMA transfer error

***I2S Exported Macros***

<code>_HAL_I2S_RESET_HANDLE_STATE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Reset I2S handle state.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: specifies the I2S Handle.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>_HAL_I2S_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Enable or disable the specified SPI peripheral (in I2S mode).</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: specifies the I2S Handle.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>_HAL_I2S_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Enable or disable the specified I2S interrupts.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: specifies the I2S Handle.</li><li><code>_INTERRUPT_</code>: specifies the interrupt source to enable or disable. This parameter can be one of the following values:<ul style="list-style-type: none"><li><code>I2S_IT_TXE</code>: Tx buffer empty interrupt enable</li><li><code>I2S_IT_RXNE</code>: RX buffer not empty interrupt enable</li><li><code>I2S_IT_ERR</code>: Error interrupt enable</li></ul></li></ul> <b>Return value:</b>
<code>_HAL_I2S_ENABLE_IT</code>	

- None

`_HAL_I2S_DISABLE_IT`  
`_HAL_I2S_GET_IT_SOURCE`

**Description:**

- Checks if the specified I2S interrupt source is enabled or disabled.

**Parameters:**

- `_HANDLE_`: specifies the I2S Handle. This parameter can be I2S where x: 1, 2, or 3 to select the I2S peripheral.
- `_INTERRUPT_`: specifies the I2S interrupt source to check. This parameter can be one of the following values:
  - `I2S_IT_TXE`: Tx buffer empty interrupt enable
  - `I2S_IT_RXNE`: RX buffer not empty interrupt enable
  - `I2S_IT_ERR`: Error interrupt enable

**Return value:**

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

`_HAL_I2S_GET_FLAG`

**Description:**

- Checks whether the specified I2S flag is set or not.

**Parameters:**

- `_HANDLE_`: specifies the I2S Handle.
- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - `I2S_FLAG_RXNE`: Receive buffer not empty flag
  - `I2S_FLAG_TXE`: Transmit buffer empty flag
  - `I2S_FLAG_UDR`: Underrun flag
  - `I2S_FLAG_OVR`: Overrun flag
  - `I2S_FLAG_FRE`: Frame error flag
  - `I2S_FLAG_CHSIDE`: Channel Side flag
  - `I2S_FLAG_BSY`: Busy flag

**Return value:**

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

`_HAL_I2S_CLEAR_OVRFAG`

**Description:**

- Clears the I2S OVR pending flag.

**Parameters:**

- `_HANDLE_`: specifies the I2S Handle.

**Return value:**

- None

`_HAL_I2S_CLEAR_UDRFLAG`

**Description:**

- Clears the I2S UDR pending flag.

**Parameters:**

- `_HANDLE_`: specifies the I2S Handle.

**Return value:**

- None

**I2S Flags Definition**

`I2S_FLAG_TXE`

`I2S_FLAG_RXNE`

`I2S_FLAG_UDR`

`I2S_FLAG_OVR`

`I2S_FLAG_FRE`

`I2S_FLAG_CHSIDE`

`I2S_FLAG_BSY`

**I2S Interrupts Definition**

`I2S_IT_TXE`

`I2S_IT_RXNE`

`I2S_IT_ERR`

**I2S Mclk Output**

`I2S_MCLKOUTPUT_ENABLE`

`I2S_MCLKOUTPUT_DISABLE`

**I2S Mode**

`I2S_MODE_SLAVE_TX`

`I2S_MODE_SLAVE_RX`

`I2S_MODE_MASTER_TX`

`I2S_MODE_MASTER_RX`

**I2S Private Macros**

`IS_I2S_CLOCKSOURCE`

`IS_I2S_MODE`

`IS_I2S_STANDARD`

`IS_I2S_DATA_FORMAT`

`IS_I2S_MCLK_OUTPUT`

`IS_I2S_AUDIO_FREQ`

`IS_I2S_CPOL`

**I2S Standard**

---

- I2S\_STANDARD\_PHILIPS
- I2S\_STANDARD\_MSB
- I2S\_STANDARD\_LSB
- I2S\_STANDARD\_PCM\_SHORT
- I2S\_STANDARD\_PCM\_LONG

## 24 HAL IRDA Generic Driver

### 24.1 IRDA Firmware driver registers structures

#### 24.1.1 IRDA\_InitTypeDef

##### 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:  

$$\text{IntegerDivider} = ((\text{PCLKx}) / (8 * (\text{hirda-} \gt \text{Init.BaudRate})))$$

$$\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{uint32\_t}) \text{ IntegerDivider})) * 8) + 0.5$$
- ***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\*](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32\_t IRDA\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [\*IRDA\\_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\*](#)

#### 24.1.2 IRDA\_HandleTypeDefDef

##### Data Fields

- *USART\_TypeDef \* Instance*
- *IRDA\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *uint16\_t TxXferSize*
- *uint16\_t TxXferCount*
- *uint8\_t \* pRxBuffPtr*

- `uint16_t RxXferSize`
- `uint16_t RxXferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `HAL_LockTypeDef Lock`
- `_IO HAL_IRDA_StateTypeDef State`
- `_IO uint32_t ErrorCode`

### Field Documentation

- `USART_TypeDef* IRDA_HandleTypeDef::Instance`
- `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 uint32_t IRDA_HandleTypeDef::ErrorCode`

## 24.2 IRDA Firmware driver API description

### 24.2.1 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

## 24.2.2 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. Depending on the frame length defined by the M bit (8-bits or 9-bits), please refer to Reference manual for possible IRDA frame formats.
- Prescaler: A pulse of width less than two and greater than one PSC period(s) may or may not be rejected. The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).
- Mode: Receiver/transmitter modes
- IrDAMode: the IrDA can operate in the Normal mode or in the Low power mode.

The HAL\_IRDA\_Init() API follows IRDA configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- [\*\*HAL\\_IRDA\\_Init\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_IRDA\\_MspDeInit\(\)\*\*](#)

### **24.2.3 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\(\)\*\*](#)

This section contains the following APIs:

- [\*\*HAL\\_IRDA\\_Transmit\(\)\*\*](#)

- [`HAL\_IRDA\_Receive\(\)`](#)
- [`HAL\_IRDA\_Transmit\_IT\(\)`](#)
- [`HAL\_IRDA\_Receive\_IT\(\)`](#)
- [`HAL\_IRDA\_Transmit\_DMA\(\)`](#)
- [`HAL\_IRDA\_Receive\_DMA\(\)`](#)
- [`HAL\_IRDA\_DMAPause\(\)`](#)
- [`HAL\_IRDA\_DMAResume\(\)`](#)
- [`HAL\_IRDA\_DMAStop\(\)`](#)
- [`HAL\_IRDA IRQHandler\(\)`](#)
- [`HAL\_IRDA\_TxCpltCallback\(\)`](#)
- [`HAL\_IRDA\_TxHalfCpltCallback\(\)`](#)
- [`HAL\_IRDA\_RxCpltCallback\(\)`](#)
- [`HAL\_IRDA\_RxHalfCpltCallback\(\)`](#)
- [`HAL\_IRDA\_ErrorCallback\(\)`](#)

#### 24.2.4 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.

This section contains the following APIs:

- [`HAL\_IRDA\_GetState\(\)`](#)
- [`HAL\_IRDA\_GetError\(\)`](#)

#### 24.2.5 HAL\_IRDA\_Init

Function Name	<code>HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)</code>
Function Description	Initializes the IRDA mode according to the specified parameters in the <code>IRDA_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> pointer to a <code>IRDA_HandleTypeDef</code> structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.6 HAL\_IRDA\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)</code>
Function Description	Deinitializes the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda:</b> pointer to a <code>IRDA_HandleTypeDef</code> structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.7 HAL\_IRDA\_MspInit



Function Name	<b>void HAL_IRDA_MspInit (IRDA_HandleTypeDef * 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>

#### 24.2.8 HAL\_IRDA\_MspDeInit

Function Name	<b>void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)</b>
Function Description	IRDA MSP DeInit.
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>

#### 24.2.9 HAL\_IRDA\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * 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> <li>• <b>Timeout:</b> Specify timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.10 HAL\_IRDA\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, 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>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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.11 HAL\_IRDA\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_IT</b>
---------------	---

**(IRDA\_HandleTypeDef \* hirda, uint8\_t \* pData, uint16\_t Size)**

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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.12 HAL\_IRDA\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_IT</b> <b>(IRDA_HandleTypeDef * hirda, 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>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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.13 HAL\_IRDA\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Transmit_DMA</b> <b>(IRDA_HandleTypeDef * hirda, 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>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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.14 HAL\_IRDA\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Receive_DMA</b> <b>(IRDA_HandleTypeDef * hirda, 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>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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the IRDA parity is enabled (PCE = 1) the data received contain the parity bit.</li> </ul>

#### 24.2.15 HAL\_IRDA\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)</b>
Function Description	Pauses the DMA Transfer.
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>• HAL status</li> </ul>

#### 24.2.16 HAL\_IRDA\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)</b>
Function Description	Resumes the DMA Transfer.
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 UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.17 HAL\_IRDA\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)</b>
Function Description	Stops the DMA Transfer.
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 UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 24.2.18 HAL\_IRDA\_IRQHandler

Function Name	<b>void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)</b>
Function Description	This function handles IRDA interrupt request.
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>

#### 24.2.19 HAL\_IRDA\_TxCpltCallback

Function Name	<b>void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Tx 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 <b>24.2.20 HAL_IRDA_TxHalfCpltCallback</b> Function Name Function Description Parameters Return values	<ul style="list-style-type: none"> <li>• None</li> </ul> <p><b>void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</b></p> <p>Tx Half Transfer completed callbacks.</p> <ul style="list-style-type: none"> <li>• <b>hirda:</b> pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.</li> </ul> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>24.2.21 HAL_IRDA_RxCpltCallback</b> Function Name Function Description Parameters Return values	<p><b>void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)</b></p> <p>Rx Transfer complete callbacks.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>24.2.22 HAL_IRDA_RxHalfCpltCallback</b> Function Name Function Description Parameters Return values	<p><b>void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</b></p> <p>Rx Half Transfer complete callbacks.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>24.2.23 HAL_IRDA_ErrorCallback</b> Function Name Function Description Parameters Return values	<p><b>void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)</b></p> <p>IRDA error callbacks.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>24.2.24 HAL_IRDA_GetState</b> Function Name Function Description Parameters	<p><b>HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)</b></p> <p>Returns the IRDA state.</p> <ul style="list-style-type: none"> <li>• <b>hirda:</b> pointer to a IRDA_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified IRDA module.

Return values • HAL state

### 24.2.25 HAL\_IRDA\_GetError

Function Name	<code>uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)</code>
Function Description	Return the IARDA 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>• IRDA Error Code</li> </ul>

## 24.3 IRDA Firmware driver defines

### 24.3.1 IRDA

#### *IRDA Error Code*

<code>HAL_IRDA_ERROR_NONE</code>	No error
<code>HAL_IRDA_ERROR_PE</code>	Parity error
<code>HAL_IRDA_ERROR_NE</code>	Noise error
<code>HAL_IRDA_ERROR_FE</code>	Frame error
<code>HAL_IRDA_ERROR_ORE</code>	Overrun error
<code>HAL_IRDA_ERROR_DMA</code>	DMA transfer error

#### *IRDA Exported Macros*

<code>_HAL_IRDA_RESET_HANDLE_STATE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Reset IRDA handle state.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>• None</li> </ul>
<code>_HAL_IRDA_FLUSH_DRREGISTER</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• Flushes the IRDA DR register.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>_HANDLE_</code>: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.</li> </ul>

[\\_\\_HAL\\_IRDA\\_GET\\_FLAG](#)**Description:**

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

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- [\\_\\_FLAG\\_\\_](#): specifies the flag to check. This parameter can be one of the following values:
  - [IRDA\\_FLAG\\_TXE](#): Transmit data register empty flag
  - [IRDA\\_FLAG\\_TC](#): Transmission Complete flag
  - [IRDA\\_FLAG\\_RXNE](#): Receive data register not empty flag
  - [IRDA\\_FLAG\\_IDLE](#): Idle Line detection flag
  - [IRDA\\_FLAG\\_ORE](#): OverRun Error flag
  - [IRDA\\_FLAG\\_NE](#): Noise Error flag
  - [IRDA\\_FLAG\\_FE](#): Framing Error flag
  - [IRDA\\_FLAG\\_PE](#): Parity Error flag

**Return value:**

- The new state of [\\_\\_FLAG\\_\\_](#) (TRUE or FALSE).

[\\_\\_HAL\\_IRDA\\_CLEAR\\_FLAG](#)**Description:**

- Clears the specified IRDA pending flag.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- [\\_\\_FLAG\\_\\_](#): specifies the flag to check. This parameter can be any combination of the following values:
  - [IRDA\\_FLAG\\_TC](#): Transmission Complete flag.
  - [IRDA\\_FLAG\\_RXNE](#): Receive data register not empty flag.

**Return value:**

- None

**Notes:**

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART\_SR register followed by a read operation to USART\_DR register. RXNE flag can be also cleared by a read to the USART\_DR register. TC flag can be also cleared by software sequence: a read operation to USART\_SR register followed by a write operation to USART\_DR register. TXE flag is cleared only by a write to the USART\_DR register.

`__HAL_IRDA_CLEAR_PEFLAG`

**Description:**

- Clear the IRDA PE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_IRDA_CLEAR_FEFLAG`

**Description:**

- Clear the IRDA FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_IRDA_CLEAR_NEFLAG`

**Description:**

- Clear the IRDA NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART

peripheral.

**Return value:**

- None

`__HAL_IRDA_CLEAR_OREFLAG`

**Description:**

- Clear the IRDA ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_IRDA_CLEAR_IDLEFLAG`

**Description:**

- Clear the IRDA IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_IRDA_ENABLE_IT`

**Description:**

- Enables or disables the specified IRDA interrupt.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- `__INTERRUPT__`: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - `IRDA_IT_TXE`: Transmit Data Register empty interrupt
  - `IRDA_IT_TC`: Transmission complete interrupt
  - `IRDA_IT_RXNE`: Receive Data register not empty interrupt
  - `IRDA_IT_IDLE`: Idle line detection interrupt
  - `IRDA_IT_PE`: Parity Error

- interrupt
  - IRDA\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

`_HAL_IRDA_DISABLE_IT`  
`_HAL_IRDA_GET_IT_SOURCE`

**Description:**

- Checks whether the specified IRDA interrupt has occurred or not.

**Parameters:**

- `_HANDLE_`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- `_IT_`: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register not empty interrupt
  - IRDA\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_ERR: Error interrupt
  - IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

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

`_HAL_IRDA_ONE_BIT_SAMPLE_ENABLE`

**Description:**

- Macro to enable the IRDA's one bit sample method.

**Parameters:**

- `_HANDLE_`: specifies the IRDA Handle.

**Return value:**

- None

`_HAL_IRDA_ONE_BIT_SAMPLE_DISABLE`

**Description:**

- Macro to disable the IRDA's one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle.

**Return value:**

- None

`__HAL_IRDA_ENABLE`**Description:**

- Enable UART/USART associated to IRDA Handle.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

`__HAL_IRDA_DISABLE`**Description:**

- Disable UART/USART associated to IRDA Handle.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

**Return value:**

- None

***IRDA Flags***

`IRDA_FLAG_TXE`  
`IRDA_FLAG_TC`  
`IRDA_FLAG_RXNE`  
`IRDA_FLAG_IDLE`  
`IRDA_FLAG_ORE`  
`IRDA_FLAG_NE`  
`IRDA_FLAG_FE`  
`IRDA_FLAG_PE`

***IRDA Interrupt Definitions***

`IRDA_IT_PE`  
`IRDA_IT_TXE`  
`IRDA_IT_TC`  
`IRDA_IT_RXNE`

IRDA\_IT\_IDLE

IRDA\_IT\_LBD

IRDA\_IT\_CTS

IRDA\_IT\_ERR

***IRDA Low Power***

IRDA\_POWERMODE\_LOWPOWER

IRDA\_POWERMODE\_NORMAL

***IRDA Transfer Mode***

IRDA\_MODE\_RX

IRDA\_MODE\_TX

IRDA\_MODE\_TX\_RX

***IRDA Parity***

IRDA\_PARITY\_NONE

IRDA\_PARITY\_EVEN

IRDA\_PARITY\_ODD

***IRDA Private Constants***

IRDA\_TIMEOUT\_VALUE

IRDA\_IT\_MASK

IRDA\_CR1\_REG\_INDEX

IRDA\_CR2\_REG\_INDEX

IRDA\_CR3\_REG\_INDEX

***IRDA Private Macros***

IS\_IRDA\_WORD\_LENGTH

IS\_IRDA\_PARITY

IS\_IRDA\_MODE

IS\_IRDA\_POWERMODE

IS\_IRDA\_BAUDRATE

IRDA\_DIV

IRDA\_DIVMANT

IRDA\_DIVFRAQ

IRDA\_BRR

***IRDA Word Length***

IRDA\_WORDLENGTH\_8B

IRDA\_WORDLENGTH\_9B

## 25 HAL IWDG Generic Driver

### 25.1 IWDG Firmware driver registers structures

#### 25.1.1 IWDG\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Reload*

##### Field Documentation

- *uint32\_t IWDG\_InitTypeDef::Prescaler*  
Select the prescaler of the IWDG. This parameter can be a value of [\*IWDG\\_Prescaler\*](#)
- *uint32\_t IWDG\_InitTypeDef::Reload*  
Specifies the IWDG down-counter reload value. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x0FFF

#### 25.1.2 IWDG\_HandleTypeDef

##### 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

### 25.2 IWDG Firmware driver API description

#### 25.2.1 IWDG Specific features

- The IWDG can be started by either software or hardware (configurable through option byte).

- The IWDG is clocked by its own dedicated Low-Speed clock (LSI) and thus stays active even if the main clock fails. Once the IWDG is started, the LSI is forced ON and cannot be disabled (LSI cannot be disabled too), and the counter starts counting down from the reset value of 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. STM32F2xx 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.

## 25.2.2 How to use this driver

If Window option is disabled

- Use IWDG using HAL\_IWDG\_Init() function to :
  - Enable write access to IWDG\_PR, IWDG\_RLR.
  - Configure the IWDG prescaler, counter reload value. This reload value will be loaded in the IWDG counter each time the counter is reloaded, then the IWDG will start counting down from this value.
- Use IWDG using HAL\_IWDG\_Start() function to:
  - Reload IWDG counter with value defined in the IWDG\_RLR register.
  - Start the IWDG, when the IWDG is used in software mode (no need to enable the LSI, it will be enabled by hardware).
- Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_IWDG\_Refresh() function.

if Window option is enabled:

- Use IWDG using HAL\_IWDG\_Start() function to enable IWDG downcounter
- Use IWDG using HAL\_IWDG\_Init() function to :
  - Enable write access to IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers.
  - Configure the IWDG prescaler, reload value and window value.
- Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_IWDG\_Refresh() function.

### IWDG HAL driver macros list

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

- \_\_HAL\_IWDG\_START: Enable the IWDG peripheral
- \_\_HAL\_IWDG\_RELOAD\_COUNTER: Reloads IWDG counter with value defined in the reload register
- \_\_HAL\_IWDG\_GET\_FLAG: Get the selected IWDG's flag status

## 25.2.3 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

This section contains the following APIs:

- [\*HAL\\_IWDG\\_Init\(\)\*](#)
- [\*HAL\\_IWDG\\_MspInit\(\)\*](#)

#### **25.2.4 IO operation functions**

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.

This section contains the following APIs:

- [\*HAL\\_IWDG\\_Start\(\)\*](#)
- [\*HAL\\_IWDG\\_Refresh\(\)\*](#)

#### **25.2.5 Peripheral State functions**

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

This section contains the following APIs:

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

#### **25.2.6 HAL\_IWDG\_Init**

Function Name	<code>HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)</code>
Function Description	Initializes the IWDG according to the specified parameters in the IWDG_InitTypeDef and creates the associated handle.
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>• HAL status</li> </ul>

#### **25.2.7 HAL\_IWDG\_MspInit**

Function Name	<code>void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)</code>
Function Description	Initializes the IWDG MSP.
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>• None</li> </ul>

#### **25.2.8 HAL\_IWDG\_Start**

Function Name	<code>HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)</code>
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>• HAL status</li> </ul>

### 25.2.9 HAL\_IWDG\_Refresh

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)</b>
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>• HAL status</li> </ul>

### 25.2.10 HAL\_IWDG\_GetState

Function Name	<b>HAL_IWDG_StateTypeDef HAL_IWDG_GetState (IWDG_HandleTypeDef * hiwdg)</b>
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>• HAL state</li> </ul>

## 25.3 IWDG Firmware driver defines

### 25.3.1 IWDG

#### *IWDG Exported Macros*

<u>_HAL_IWDG_RESET_HANDLE_STATE</u>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Reset IWDG handle state.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <u>_HANDLE__</u>: IWDG handle.</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>
<u>_HAL_IWDG_START</u>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Enables the IWDG peripheral.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <u>_HANDLE__</u>: IWDG handle</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>
<u>_HAL_IWDG_RELOAD_COUNTER</u>	<b>Description:</b>

- Reloads IWDG counter with value defined in the reload register (write access to IWDG\_PR and IWDG\_RLR registers disabled).

**Parameters:**

- `__HANDLE__`: IWDG handle

**Return value:**

- None

**`_HAL_IWDG_GET_FLAG`**

- Gets the selected IWDG's flag status.

**Parameters:**

- `__HANDLE__`: IWDG handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `IWDG_FLAG_PVU`: Watchdog counter reload value update flag
  - `IWDG_FLAG_RVU`: Watchdog counter prescaler value flag

**Return value:**

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

***IWDG Flag definition***

`IWDG_FLAG_PVU`      Watchdog counter prescaler value update Flag

`IWDG_FLAG_RVU`      Watchdog counter reload value update Flag

***IWDG Prescaler***

`IWDG_PRESCALER_4`      IWDG prescaler set to 4

`IWDG_PRESCALER_8`      IWDG prescaler set to 8

`IWDG_PRESCALER_16`      IWDG prescaler set to 16

`IWDG_PRESCALER_32`      IWDG prescaler set to 32

`IWDG_PRESCALER_64`      IWDG prescaler set to 64

`IWDG_PRESCALER_128`      IWDG prescaler set to 128

`IWDG_PRESCALER_256`      IWDG prescaler set to 256

***IWDG Private Constants***

`IWDG_TIMEOUT_FLAG`

***IWDG Private Macros***

`IWDG_ENABLE_WRITE_ACCESS`      **Description:**

- Enables write access to IWDG\_PR and IWDG\_RLR registers.

**Parameters:**

- `__HANDLE__`: IWDG handle

**Return value:**

- None

**IWDG\_DISABLE\_WRITE\_ACCESS****Description:**

- Disables write access to IWDG\_PR and IWDG\_RLR registers.

**Parameters:**

- `__HANDLE__`: IWDG handle

**Return value:**

- None

**IS\_IWDG\_PRESCALER****IS\_IWDG\_RELOAD*****IWDG Registers BitMask***

<code>IWDG_KEY_RELOAD</code>	IWDG Reload Counter Enable
<code>IWDG_KEY_ENABLE</code>	IWDG Peripheral Enable
<code>IWDG_KEY_WRITE_ACCESS_ENABLE</code>	IWDG KR Write Access Enable
<code>IWDG_KEY_WRITE_ACCESS_DISABLE</code>	IWDG KR Write Access Disable

## 26 HAL NAND Generic Driver

### 26.1 NAND Firmware driver registers structures

#### 26.1.1 NAND\_IDTypeDef

##### Data Fields

- *uint8\_t Maker\_Id*
- *uint8\_t Device\_Id*
- *uint8\_t Third\_Id*
- *uint8\_t Fourth\_Id*

##### Field Documentation

- *uint8\_t NAND\_IDTypeDef::Maker\_Id*
- *uint8\_t NAND\_IDTypeDef::Device\_Id*
- *uint8\_t NAND\_IDTypeDef::Third\_Id*
- *uint8\_t NAND\_IDTypeDef::Fourth\_Id*

#### 26.1.2 NAND\_AddressTypeDef

##### 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

#### 26.1.3 NAND\_InfoTypeDef

##### Data Fields

- *uint32\_t PageSize*
- *uint32\_t SpareAreaSize*
- *uint32\_t BlockSize*
- *uint32\_t BlockNbr*

- *uint32\_t ZoneSize*

#### Field Documentation

- *uint32\_t NAND\_InfoTypeDef::PageSize*  
NAND memory page (without spare area) size measured in K. bytes
- *uint32\_t NAND\_InfoTypeDef::SpareAreaSize*  
NAND memory spare area size measured in K. bytes
- *uint32\_t NAND\_InfoTypeDef::BlockSize*  
NAND memory block size number of pages
- *uint32\_t NAND\_InfoTypeDef::BlockNbr*  
NAND memory number of blocks
- *uint32\_t NAND\_InfoTypeDef::ZoneSize*  
NAND memory zone size measured in number of blocks

### 26.1.4 NAND\_HandleTypeDef

#### Data Fields

- *FSMC\_NAND\_TypeDef \* Instance*
- *FSMC\_NAND\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_NAND\_StateTypeDef State*
- *NAND\_InfoTypeDef Info*

#### Field Documentation

- *FSMC\_NAND\_TypeDef\* NAND\_HandleTypeDef::Instance*  
Register base address
- *FSMC\_NAND\_InitTypeDef NAND\_HandleTypeDef::Init*  
NAND device control configuration parameters
- *HAL\_LockTypeDef NAND\_HandleTypeDef::Lock*  
NAND locking object
- *\_\_IO HAL\_NAND\_StateTypeDef NAND\_HandleTypeDef::State*  
NAND device access state
- *NAND\_InfoTypeDef NAND\_HandleTypeDef::Info*  
NAND characteristic information structure

## 26.2 NAND Firmware driver API description

### 26.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FSMC 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.

## 26.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

This section contains the following APIs:

- [\*\*HAL\\_NAND\\_Init\(\)\*\*](#)
- [\*\*HAL\\_NAND\\_DelInit\(\)\*\*](#)
- [\*\*HAL\\_NAND\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_NAND\\_MspDelInit\(\)\*\*](#)
- [\*\*HAL\\_NAND\\_IRQHandler\(\)\*\*](#)
- [\*\*HAL\\_NAND\\_ITCallback\(\)\*\*](#)

## 26.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

This section contains the following APIs:

- [\*\*HAL\\_NAND\\_Read\\_ID\(\)\*\*](#)
- [\*\*HAL\\_NAND\\_Reset\(\)\*\*](#)
- [\*\*HAL\\_NAND\\_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\(\)\*\*](#)

## 26.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

This section contains the following APIs:

- [\*HAL\\_NAND\\_ECC\\_Enable\(\)\*](#)
- [\*HAL\\_NAND\\_ECC\\_Disable\(\)\*](#)
- [\*HAL\\_NAND\\_GetECC\(\)\*](#)

## 26.2.5 NAND State functions

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

This section contains the following APIs:

- [\*HAL\\_NAND\\_GetState\(\)\*](#)
- [\*HAL\\_NAND\\_Read\\_Status\(\)\*](#)

## 26.2.6 HAL\_NAND\_Init

Function Name	<code>HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef *hnand, FSMC_NAND_PCC_TimingTypeDef *ComSpace_Timing, FSMC_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 NAND_HandleTypeDef 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	HAL status

## 26.2.7 HAL\_NAND\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_NAND_DelInit (NAND_HandleTypeDef *hnand)</code>
Function Description	Perform NAND memory De-Initialization sequence.
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

HAL status

## 26.2.8 HAL\_NAND\_MspInit

Function Name	<code>void HAL_NAND_MspInit (NAND_HandleTypeDef *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	None

### 26.2.9 HAL\_NAND\_MspDeInit

Function Name	<b>void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hhand)</b>
Function Description	NAND MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhand:</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>

### 26.2.10 HAL\_NAND\_IRQHandler

Function Name	<b>void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hhand)</b>
Function Description	This function handles NAND device interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhand:</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>• HAL status</li> </ul>

### 26.2.11 HAL\_NAND\_ITCallback

Function Name	<b>void HAL_NAND_ITCallback (NAND_HandleTypeDef * hhand)</b>
Function Description	NAND interrupt feature callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhand:</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>

### 26.2.12 HAL\_NAND\_Read\_ID

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hhand, NAND_IDTypeDef * pNAND_ID)</b>
Function Description	Read the NAND memory electronic signature.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhand:</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>• HAL status</li> </ul>

### 26.2.13 HAL\_NAND\_Reset

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hhand)</b>
Function Description	NAND memory reset.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhand:</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>• HAL status</li> </ul>

### 26.2.14 HAL\_NAND\_Read\_Page

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Read_Page (NAND_HandleTypeDef * hndl, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)</b>
Function Description	Read Page(s) from NAND memory block.
Parameters	<ul style="list-style-type: none"> <li><b>hndl:</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>HAL status</li> </ul>

### 26.2.15 HAL\_NAND\_Write\_Page

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Write_Page (NAND_HandleTypeDef * hndl, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)</b>
Function Description	Write Page(s) to NAND memory block.
Parameters	<ul style="list-style-type: none"> <li><b>hndl:</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>HAL status</li> </ul>

### 26.2.16 HAL\_NAND\_Read\_SpareArea

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Read_SpareArea (NAND_HandleTypeDef * hndl, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)</b>
Function Description	Read Spare area(s) from NAND memory.
Parameters	<ul style="list-style-type: none"> <li><b>hndl:</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>HAL status</li> </ul>

### 26.2.17 HAL\_NAND\_Write\_SpareArea

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Write_SpareArea (NAND_HandleTypeDef * hndl, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)</b>
Function Description	Write Spare area(s) to NAND memory.
Parameters	<ul style="list-style-type: none"> <li><b>hndl:</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</li> </ul>

block

Return values

- HAL status

### 26.2.18 HAL\_NAND\_Erase\_Block

Function Name      **HAL\_StatusTypeDef HAL\_NAND\_Erase\_Block  
(NAND\_HandleTypeDef \* hndl, NAND\_AddressTypeDef \*  
pAddress)**

Function Description      NAND memory Block erase.

Parameters

- **hndl:** pointer to a NAND\_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** : pointer to NAND address structure

Return values

- HAL status

### 26.2.19 HAL\_NAND\_Read\_Status

Function Name      **uint32\_t HAL\_NAND\_Read\_Status (NAND\_HandleTypeDef \*  
hndl)**

Function Description      NAND memory read status.

Parameters

- **hndl:** pointer to a NAND\_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- NAND status

### 26.2.20 HAL\_NAND\_Address\_Inc

Function Name      **uint32\_t HAL\_NAND\_Address\_Inc (NAND\_HandleTypeDef \*  
hndl, NAND\_AddressTypeDef \* pAddress)**

Function Description      Increment the NAND memory address.

Parameters

- **hndl:** pointer to a NAND\_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- The new status of the increment address operation. It can be:  
**NAND\_VALID\_ADDRESS:** When the new address is valid  
**NAND\_INVALID\_ADDRESS:** When the new address is invalid address

### 26.2.21 HAL\_NAND\_ECC\_Enable

Function Name      **HAL\_StatusTypeDef HAL\_NAND\_ECC\_Enable  
(NAND\_HandleTypeDef \* hndl)**

Function Description      Enables dynamically NAND ECC feature.

Parameters

- **hndl:** pointer to a NAND\_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- HAL status

### 26.2.22 HAL\_NAND\_ECC\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hndl)</b>
Function Description	Disables dynamically FSMC_NAND ECC feature.
Parameters	<ul style="list-style-type: none"> <li><b>hndl</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>HAL status</li> </ul>

### 26.2.23 HAL\_NAND\_GetECC

Function Name	<b>HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hndl, uint32_t * ECCval, uint32_t Timeout)</b>
Function Description	Disables dynamically NAND ECC feature.
Parameters	<ul style="list-style-type: none"> <li><b>hndl</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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 26.2.24 HAL\_NAND\_GetState

Function Name	<b>HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hndl)</b>
Function Description	return the NAND state
Parameters	<ul style="list-style-type: none"> <li><b>hndl</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>HAL state</li> </ul>

### 26.2.25 HAL\_NAND\_Read\_Status

Function Name	<b>uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hndl)</b>
Function Description	NAND memory read status.
Parameters	<ul style="list-style-type: none"> <li><b>hndl</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>NAND status</li> </ul>

## 26.3 NAND Firmware driver defines

### 26.3.1 NAND

#### *NAND Exported Macros*

**\_HAL\_NAND\_RESET\_HANDLE\_STATE**   **Description:**

- Reset NAND handle state.

**Parameters:**

- \_HANDLE\_**: specifies the NAND

handle.

**Return value:**

- None

**NAND Private Constants**

NAND\_DEVICE1  
NAND\_DEVICE2  
NAND\_WRITE\_TIMEOUT  
CMD\_AREA  
ADDR\_AREA  
NAND\_CMD\_AREA\_A  
NAND\_CMD\_AREA\_B  
NAND\_CMD\_AREA\_C  
NAND\_CMD\_AREA\_TRUE1  
NAND\_CMD\_WRITE0  
NAND\_CMD\_WRITE\_TRUE1  
NAND\_CMD\_ERASE0  
NAND\_CMD\_ERASE1  
NAND\_CMD\_READID  
NAND\_CMD\_STATUS  
NAND\_CMD\_LOCK\_STATUS  
NAND\_CMD\_RESET  
NAND\_VALID\_ADDRESS  
NAND\_INVALID\_ADDRESS  
NAND\_TIMEOUT\_ERROR  
NAND\_BUSY  
NAND\_ERROR  
NAND\_READY

**NAND Private Macros**

ARRAY_ADDRESS	<b>Description:</b>
	<ul style="list-style-type: none"><li>• NAND memory address computation.</li></ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"><li>• <u>__ADDRESS__</u>: NAND memory address.</li><li>• <u>__HANDLE__</u>: NAND handle.</li></ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"><li>• NAND: Raw address value</li></ul>
ADDR_1ST_CYCLE	<b>Description:</b>

- NAND memory address cycling.

**Parameters:**

- ADDRESS: NAND memory address.

**Return value:**

- NAND: address cycling value.

ADDR\_2ND\_CYCLE

ADDR\_3RD\_CYCLE

ADDR\_4TH\_CYCLE

## 27 HAL NOR Generic Driver

### 27.1 NOR Firmware driver registers structures

#### 27.1.1 NOR\_IDTypeDef

##### Data Fields

- *uint16\_t Manufacturer\_Code*
- *uint16\_t Device\_Code1*
- *uint16\_t Device\_Code2*
- *uint16\_t Device\_Code3*

##### Field Documentation

- ***uint16\_t NOR\_IDTypeDef::Manufacturer\_Code***  
Defines the device's manufacturer code used to identify the memory
- ***uint16\_t NOR\_IDTypeDef::Device\_Code1***
- ***uint16\_t NOR\_IDTypeDef::Device\_Code2***
- ***uint16\_t NOR\_IDTypeDef::Device\_Code3***  
Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set. They can also be accessed by issuing an Auto Select command

#### 27.1.2 NOR\_CFITypeDef

##### Data Fields

- *uint16\_t CFI\_1*
- *uint16\_t CFI\_2*
- *uint16\_t CFI\_3*
- *uint16\_t CFI\_4*

##### Field Documentation

- ***uint16\_t NOR\_CFITypeDef::CFI\_1***  
< Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- ***uint16\_t NOR\_CFITypeDef::CFI\_2***
- ***uint16\_t NOR\_CFITypeDef::CFI\_3***
- ***uint16\_t NOR\_CFITypeDef::CFI\_4***

#### 27.1.3 NOR\_HandleTypeDef

**Data Fields**

- ***FSMC\_NORSRAM\_TypeDef \* Instance***
- ***FSMC\_NORSRAM\_EXTENDED\_TypeDef \* Extended***
- ***FSMC\_NORSRAM\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_NOR\_StateTypeDef State***

**Field Documentation**

- ***FSMC\_NORSRAM\_TypeDef\* NOR\_HandleTypeDef::Instance***  
Register base address
- ***FSMC\_NORSRAM\_EXTENDED\_TypeDef\* NOR\_HandleTypeDef::Extended***  
Extended mode register base address
- ***FSMC\_NORSRAM\_InitTypeDef NOR\_HandleTypeDef::Init***  
NOR device control configuration parameters
- ***HAL\_LockTypeDef NOR\_HandleTypeDef::Lock***  
NOR locking object
- ***\_\_IO HAL\_NOR\_StateTypeDef NOR\_HandleTypeDef::State***  
NOR device access state

## 27.2 NOR Firmware driver API description

### 27.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FSMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL\_NOR\_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL\_NOR\_Read\_ID(). The read information is stored in the NOR\_ID\_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL\_NOR\_Read(), HAL\_NOR\_Program().
- Perform NOR flash erase block/chip operations using the functions HAL\_NOR\_Erase\_Block() and HAL\_NOR\_Erase\_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL\_NOR\_Read\_CFI(). The read information is stored in the NOR\_CFI\_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL\_NOR\_WriteOperation\_Enable() / HAL\_NOR\_WriteOperation\_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL\_NOR\_GetState()



This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

### NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- NOR\_WRITE : NOR memory write data to specified address

### 27.2.2 NOR Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- [\*HAL\\_NOR\\_Init\(\)\*](#)
- [\*HAL\\_NOR\\_DelInit\(\)\*](#)
- [\*HAL\\_NOR\\_MspInit\(\)\*](#)
- [\*HAL\\_NOR\\_MspDelInit\(\)\*](#)
- [\*HAL\\_NOR\\_MspWait\(\)\*](#)

### 27.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

This section contains the following APIs:

- [\*HAL\\_NOR\\_Read\\_ID\(\)\*](#)
- [\*HAL\\_NOR\\_ReturnToReadMode\(\)\*](#)
- [\*HAL\\_NOR\\_Read\(\)\*](#)
- [\*HAL\\_NOR\\_Program\(\)\*](#)
- [\*HAL\\_NOR\\_ReadBuffer\(\)\*](#)
- [\*HAL\\_NOR\\_ProgramBuffer\(\)\*](#)
- [\*HAL\\_NOR\\_Erase\\_Block\(\)\*](#)
- [\*HAL\\_NOR\\_Erase\\_Chip\(\)\*](#)
- [\*HAL\\_NOR\\_Read\\_CFI\(\)\*](#)

### 27.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

This section contains the following APIs:

- [\*HAL\\_NOR\\_WriteOperation\\_Enable\(\)\*](#)
- [\*HAL\\_NOR\\_WriteOperation\\_Disable\(\)\*](#)

### 27.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

This section contains the following APIs:

- [\*HAL\\_NOR\\_GetState\(\)\*](#)
- [\*HAL\\_NOR\\_GetStatus\(\)\*](#)

### 27.2.6 HAL\_NOR\_Init

Function Name	<code>HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_NORSRAM_TimingTypeDef * ExtTiming)</code>
---------------	---

Function Description	Perform the NOR memory Initialization sequence.
----------------------	---

Parameters	<ul style="list-style-type: none"> <li><b>hnor:</b> pointer to the NOR handle</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>HAL status</li> </ul>

### 27.2.7 HAL\_NOR\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)</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>HAL status</li> </ul>

### 27.2.8 HAL\_NOR\_MspInit

Function Name	<b>void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)</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>

### 27.2.9 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>

### 27.2.10 HAL\_NOR\_MspWait

Function Name	<b>void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)</b>
Function Description	NOR BSP Wait for 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>

### 27.2.11 HAL\_NOR\_Read\_ID

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)</b>
Function Description	Read NOR flash IDs.
Parameters	<ul style="list-style-type: none"> <li><b>hnor:</b> pointer to the NOR handle</li> </ul>

- **pNOR\_ID:** : pointer to NOR ID structure
- Return values HAL status

### 27.2.12 HAL\_NOR\_ReturnToReadMode

Function Name	<b>HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)</b>
Function Description	Returns the NOR memory to Read mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor:</b> pointer to the NOR handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 27.2.13 HAL\_NOR\_Read

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)</b>
Function Description	Read data from NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor:</b> pointer to the NOR handle</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>• HAL status</li> </ul>

### 27.2.14 HAL\_NOR\_Program

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)</b>
Function Description	Program data to NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor:</b> pointer to the NOR handle</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>• HAL status</li> </ul>

### 27.2.15 HAL\_NOR\_ReadBuffer

Function Name	<b>HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)</b>
Function Description	Reads a half-word buffer from the NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor:</b> pointer to the NOR handle</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>• HAL status</li> </ul>

### 27.2.16 HAL\_NOR\_ProgramBuffer

Function Name	<b>HAL_StatusTypeDef HAL_NOR_ProgramBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)</b>
Function Description	Writes a half-word buffer to the NOR memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor:</b> pointer to the NOR handle</li> <li>• <b>uwAddress:</b> NOR memory internal start write address</li> <li>• <b>pData:</b> pointer to source data buffer.</li> <li>• <b>uwBufferSize:</b> Size of the buffer to write</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 27.2.17 HAL\_NOR\_Erase\_Block

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Erase_Block (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 the NOR handle</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>• HAL status</li> </ul>

### 27.2.18 HAL\_NOR\_Erase\_Chip

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Erase_Chip (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 the NOR handle</li> <li>• <b>Address:</b> : Device address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 27.2.19 HAL\_NOR\_Read\_CFI

Function Name	<b>HAL_StatusTypeDef HAL_NOR_Read_CFI (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 the NOR handle</li> <li>• <b>pNOR_CFI:</b> : pointer to NOR CFI IDs structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 27.2.20 HAL\_NOR\_WriteOperation\_Enable

Function Name	<b>HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)</b>
Function Description	Enables dynamically NOR write operation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor:</b> pointer to the NOR handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 27.2.21 HAL\_NOR\_WriteOperation\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)</b>
Function Description	Disables dynamically NOR write operation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor:</b> pointer to the NOR handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 27.2.22 HAL\_NOR\_GetState

Function Name	<b>HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)</b>
Function Description	return the NOR controller state
Parameters	<ul style="list-style-type: none"> <li>• <b>hnor:</b> pointer to the NOR handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• NOR controller state</li> </ul>

### 27.2.23 HAL\_NOR\_GetStatus

Function Name	<b>HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (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 the NOR handle</li> <li>• <b>Address:</b> Device address</li> <li>• <b>Timeout:</b> NOR programming Timeout</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• NOR_Status The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT</li> </ul>

## 27.3 NOR Firmware driver defines

### 27.3.1 NOR

#### *NOR Exported Macros*

<u>_HAL_NOR_RESET_HANDLE_STATE</u>	<b>Description:</b>
	<ul style="list-style-type: none"> <li>• Reset NOR handle state.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <u>_HANDLE_</u>: specifies the NOR handle.</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### *NOR Private Constants*

MC\_ADDRESS

DEVICE\_CODE1\_ADDR

DEVICE\_CODE2\_ADDR  
DEVICE\_CODE3\_ADDR  
CFI1\_ADDRESS  
CFI2\_ADDRESS  
CFI3\_ADDRESS  
CFI4\_ADDRESS  
NOR\_TMEOUT  
NOR\_MEMORY\_8B  
NOR\_MEMORY\_16B  
NOR\_MEMORY\_ADRESS1  
NOR\_MEMORY\_ADRESS2  
NOR\_MEMORY\_ADRESS3  
NOR\_MEMORY\_ADRESS4

**NOR Private Defines**

NOR\_CMD\_ADDRESS\_FIRST  
NOR\_CMD\_ADDRESS\_FIRST\_CFI  
NOR\_CMD\_ADDRESS\_SECOND  
NOR\_CMD\_ADDRESS\_THIRD  
NOR\_CMD\_ADDRESS\_FOURTH  
NOR\_CMD\_ADDRESS\_FIFTH  
NOR\_CMD\_ADDRESS\_SIXTH  
NOR\_CMD\_DATA\_READ\_RESET  
NOR\_CMD\_DATA\_FIRST  
NOR\_CMD\_DATA\_SECOND  
NOR\_CMD\_DATA\_AUTO\_SELECT  
NOR\_CMD\_DATA\_PROGRAM  
NOR\_CMD\_DATA\_CHIP\_BLOCK\_ERASE\_THIRD  
NOR\_CMD\_DATA\_CHIP\_BLOCK\_ERASE\_FOURTH  
NOR\_CMD\_DATA\_CHIP\_BLOCK\_ERASE\_FIFTH  
NOR\_CMD\_DATA\_CHIP\_ERASE  
NOR\_CMD\_DATA\_CFI  
NOR\_CMD\_DATA\_BUFFER\_AND\_PROG  
NOR\_CMD\_DATA\_BUFFER\_AND\_PROG\_CONFIRM  
NOR\_CMD\_DATA\_BLOCK\_ERASE  
NOR\_MASK\_STATUS\_DQ5  
NOR\_MASK\_STATUS\_DQ6

**NOR Private Macros**

**NOR\_ADDR\_SHIFT****Description:**

- NOR memory address shifting.

**Parameters:**

- NOR\_ADDRESS: NOR base address
- NOR\_MEMORY\_WIDTH: NOR memory width
- ADDRESS: NOR memory address

**Return value:**

- NOR: shifted address value

**NOR\_WRITE****Description:**

- NOR memory write data to specified address.

**Parameters:**

- ADDRESS: NOR memory address
- DATA: Data to write

**Return value:**

- None

## 28 HAL PCCARD Generic Driver

### 28.1 PCCARD Firmware driver registers structures

#### 28.1.1 PCCARD\_HandleTypeDef

##### Data Fields

- *FSMC\_PCCARD\_TypeDef \* Instance*
- *FSMC\_PCCARD\_InitTypeDef Init*
- *\_IO HAL\_PCCARD\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*

##### Field Documentation

- ***FSMC\_PCCARD\_TypeDef\* PCCARD\_HandleTypeDef::Instance***  
Register base address for PCCARD device
- ***FSMC\_PCCARD\_InitTypeDef PCCARD\_HandleTypeDef::Init***  
PCCARD device control configuration parameters
- ***\_IO HAL\_PCCARD\_StateTypeDef PCCARD\_HandleTypeDef::State***  
PCCARD device access state
- ***HAL\_LockTypeDef PCCARD\_HandleTypeDef::Lock***  
PCCARD Lock

### 28.2 PCCARD Firmware driver API description

#### 28.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control PCCARD/compact flash memories. It uses the FSMC layer functions to interface with PCCARD devices. This driver is used for:

- PCCARD/Compact Flash memory configuration sequence using the function HAL\_PCCARD\_Init()/HAL\_CF\_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\_PCCARD\_Read\_ID()/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\_PCCARD\_Read\_Sector()/ HAL\_PCCARD\_Write\_Sector() - HAL\_CF\_Read\_Sector()/HAL\_CF\_Write\_Sector(), to read/write sector.
- Perform PCCARD/Compact Flash Reset chip operation using the function HAL\_PCCARD\_Reset()/HAL\_CF\_Reset.
- Perform PCCARD/Compact Flash erase sector operation using the function HAL\_PCCARD\_Erase\_Sector()/HAL\_CF\_Erase\_Sector.
- Read the PCCARD/Compact Flash status operation using the function HAL\_PCCARD\_ReadStatus()/HAL\_CF\_ReadStatus().
- You can monitor the PCCARD/Compact Flash device HAL state by calling the function HAL\_PCCARD\_GetState()/HAL\_CF\_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.

## 28.2.2 PCCARD Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the PCCARD memory

This section contains the following APIs:

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

## 28.2.3 PCCARD Input and Output functions

This section provides functions allowing to use and control the PCCARD memory

This section contains the following APIs:

- [\*HAL\\_PCCARD\\_Read\\_ID\(\)\*](#)
- [\*HAL\\_PCCARD\\_Read\\_Sector\(\)\*](#)
- [\*HAL\\_PCCARD\\_Write\\_Sector\(\)\*](#)
- [\*HAL\\_PCCARD\\_Erase\\_Sector\(\)\*](#)
- [\*HAL\\_PCCARD\\_Reset\(\)\*](#)
- [\*HAL\\_PCCARD\\_IRQHandler\(\)\*](#)
- [\*HAL\\_PCCARD\\_ITCallback\(\)\*](#)

## 28.2.4 PCCARD State functions

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

This section contains the following APIs:

- [\*HAL\\_PCCARD\\_GetState\(\)\*](#)
- [\*HAL\\_PCCARD\\_GetStatus\(\)\*](#)
- [\*HAL\\_PCCARD\\_ReadStatus\(\)\*](#)

## 28.2.5 HAL\_PCCARD\_Init

Function Name	<code>HAL_StatusTypeDef HAL_PCCARD_Init (PCCARD_HandleTypeDef * hpccard, FSMC_NAND_PCC_TimingTypeDef * ComSpaceTiming, FSMC_NAND_PCC_TimingTypeDef * AttSpaceTiming, FSMC_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	• HAL status

### 28.2.6 HAL\_PCCARD\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_PCCARD_DelInit (PCCARD_HandleTypeDef * hpccard)</code>
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>• HAL status</li> </ul>

### 28.2.7 HAL\_PCCARD\_MspInit

Function Name	<code>void HAL_PCCARD_MspInit (PCCARD_HandleTypeDef * hpccard)</code>
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>

### 28.2.8 HAL\_PCCARD\_MspDelInit

Function Name	<code>void HAL_PCCARD_MspDelInit (PCCARD_HandleTypeDef * 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>

### 28.2.9 HAL\_PCCARD\_Read\_ID

Function Name	<code>HAL_StatusTypeDef HAL_PCCARD_Read_ID (PCCARD_HandleTypeDef * 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>• HAL status</li> </ul>

### 28.2.10 HAL\_PCCARD\_Read\_Sector

Function Name	<code>HAL_StatusTypeDef HAL_PCCARD_Read_Sector (PCCARD_HandleTypeDef * 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 PCCARD status</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 28.2.11 HAL\_PCCARD\_Write\_Sector

Function Name	<b>HAL_StatusTypeDef HAL_PCCARD_Write_Sector(PCCARD_HandleTypeDef * hpccard, uint16_t * pBuffer, uint16_t SectorAddress, uint8_t * pStatus)</b>
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 PCCARD status</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 28.2.12 HAL\_PCCARD\_Erase\_Sector

Function Name	<b>HAL_StatusTypeDef HAL_PCCARD_Erase_Sector(PCCARD_HandleTypeDef * hpccard, uint16_t SectorAddress, uint8_t * pStatus)</b>
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> <li>• <b>pStatus:</b> pointer to PCCARD status</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 28.2.13 HAL\_PCCARD\_Reset

Function Name	<b>HAL_StatusTypeDef HAL_PCCARD_Reset(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>• HAL status</li> </ul>

### 28.2.14 HAL\_PCCARD\_IRQHandler

Function Name	<b>void HAL_PCCARD_IRQHandler (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>• HAL status</li> </ul>

### 28.2.15 HAL\_PCCARD\_ITCallback

Function Name	<b>void HAL_PCCARD_ITCallback (PCCARD_HandleTypeDef * 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>

### 28.2.16 HAL\_PCCARD\_GetState

Function Name	<b>HAL_PCCARD_StateTypeDef HAL_PCCARD_GetState (PCCARD_HandleTypeDef * 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>• HAL state</li> </ul>

### 28.2.17 HAL\_PCCARD\_GetStatus

Function Name	<b>HAL_PCCARD_StatusTypeDef HAL_PCCARD_GetStatus (PCCARD_HandleTypeDef * 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>• New status of the PCCARD operation. This parameter can be: CompactFlash_TIMEOUT_ERROR: when the previous operation generate a Timeout errorCompactFlash_READY: when memory is ready for the next operation</li> </ul>

### 28.2.18 HAL\_PCCARD\_ReadStatus

Function Name	<b>HAL_PCCARD_StatusTypeDef HAL_PCCARD_ReadStatus (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>• The status of the Compact Flash memory. This parameter can be: CompactFlash_BUSY: when memory is busyCompactFlash_READY: when memory is ready for the next operationCompactFlash_ERROR: when the previous operation generates error</li> </ul>

## 28.3 PCCARD Firmware driver defines

### 28.3.1 PCCARD

#### PCCARD Exported Macros

`_HAL_PCCARD_RESET_HANDLE_STATE` **Description:**

- Reset PCCARD handle state.

#### Parameters:

- `_HANDLE_`: specifies the PCCARD handle.

#### Return value:

- None

#### PCCARD Private Constants

`PCCARD_DEVICE_ADDRESS`

`PCCARD_ATTRIBUTE_SPACE_ADDRESS`

`PCCARD_COMMON_SPACE_ADDRESS`

`PCCARD_IO_SPACE_ADDRESS`

`PCCARD_IO_SPACE_PRIMARY_ADDR`

`ATA_DATA`

`ATA_SECTOR_COUNT`

`ATA_SECTOR_NUMBER`

`ATA_CYLINDER_LOW`

`ATA_CYLINDER_HIGH`

`ATA_CARD_HEAD`

`ATA_STATUS_CMD`

`ATA_STATUS_CMD_ALTERNATE`

`ATA_COMMON_DATA_AREA`

`ATA_CARD_CONFIGURATION`

`ATA_READ_SECTOR_CMD`

`ATA_WRITE_SECTOR_CMD`

`ATA_ERASE_SECTOR_CMD`

ATA\_IDENTIFY\_CMD  
PCCARD\_TIMEOUT\_ERROR  
PCCARD\_BUSY  
PCCARD\_PROGR  
PCCARD\_READY  
PCCARD\_SECTOR\_SIZE  
**PCCARD Private Defines**  
PCCARD\_TIMEOUT\_READ\_ID  
PCCARD\_TIMEOUT\_READ\_WRITE\_SECTOR  
PCCARD\_TIMEOUT\_ERASE\_SECTOR  
PCCARD\_TIMEOUT\_STATUS  
PCCARD\_STATUS\_OK  
PCCARD\_STATUS\_WRITE\_OK

## 29 HAL PCD Generic Driver

### 29.1 PCD Firmware driver registers structures

#### 29.1.1 PCD\_HandleTypeDef

##### Data Fields

- *PCD\_TypeDef \* Instance*
- *PCD\_InitTypeDef Init*
- *PCD\_EPTTypeDef IN\_ep*
- *PCD\_EPTTypeDef OUT\_ep*
- *HAL\_LockTypeDef Lock*
- *\_\_IO PCD\_StateTypeDef State*
- *uint32\_t Setup*
- *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

## 29.2 PCD Firmware driver API description

### 29.2.1 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 PCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL\_PCD\_MspInit() API:
  - a. Enable the PCD/USB Low Level interface clock using
    - *\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_ENABLE();*

- `__HAL_RCC_USB_OTG_HS_CLK_ENABLE();` (For High Speed Mode)
- b. Initialize the related GPIO clocks
- c. Configure PCD pin-out
- d. Configure PCD NVIC interrupt
- 5. Associate the Upper USB device stack to the HAL PCD Driver:
  - a. `hpcd.pData = pdev;`
- 6. Enable PCD transmission and reception:
  - a. `HAL_PCD_Start();`

## 29.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- `HAL_PCD_Init()`
- `HAL_PCD_DelInit()`
- `HAL_PCD_MspInit()`
- `HAL_PCD_MspDelInit()`

## 29.2.3 IO operation functions

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

This section contains the following APIs:

- `HAL_PCD_Start()`
- `HAL_PCD_Stop()`
- `HAL_PCD_IRQHandler()`
- `HAL_PCD_DataOutStageCallback()`
- `HAL_PCD_DataInStageCallback()`
- `HAL_PCD_SetupStageCallback()`
- `HAL_PCD_SOFCallback()`
- `HAL_PCD_ResetCallback()`
- `HAL_PCD_SuspendCallback()`
- `HAL_PCD_ResumeCallback()`
- `HAL_PCD_ISOOUTIncompleteCallback()`
- `HAL_PCD_ISOINIncompleteCallback()`
- `HAL_PCD_ConnectCallback()`
- `HAL_PCD_DisconnectCallback()`

## 29.2.4 Peripheral Control functions

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

This section contains the following APIs:

- `HAL_PCD_DevConnect()`
- `HAL_PCD_DevDisconnect()`
- `HAL_PCD_SetAddress()`
- `HAL_PCD_EP_Open()`
- `HAL_PCD_EP_Close()`
- `HAL_PCD_EP_Receive()`
- `HAL_PCD_EP_GetRxCount()`
- `HAL_PCD_EP_Transmit()`
- `HAL_PCD_EP_SetStall()`
- `HAL_PCD_EP_ClrStall()`
- `HAL_PCD_EP_Flush()`

- [\*HAL\\_PCD\\_ActivateRemoteWakeUp\(\)\*](#)
- [\*HAL\\_PCD\\_DeActivateRemoteWakeUp\(\)\*](#)

### 29.2.5 Peripheral State functions

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

This section contains the following APIs:

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

### 29.2.6 HAL\_PCD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)</b>
Function Description	Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and initialize 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>• HAL status</li> </ul>

### 29.2.7 HAL\_PCD\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DelInit (PCD_HandleTypeDef * 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>• HAL status</li> </ul>

### 29.2.8 HAL\_PCD\_MspInit

Function Name	<b>void HAL_PCD_MspInit (PCD_HandleTypeDef * 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>

### 29.2.9 HAL\_PCD\_MspDelInit

Function Name	<b>void HAL_PCD_MspDelInit (PCD_HandleTypeDef * 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>

### 29.2.10 HAL\_PCD\_Start

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * 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>• HAL status</li></ul>
---------------	--

### 29.2.11 HAL\_PCD\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * 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>• HAL status</li></ul>

### 29.2.12 HAL\_PCD\_IRQHandler

Function Name	<b>void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)</b>
Function Description	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>• HAL status</li></ul>

### 29.2.13 HAL\_PCD\_DataOutStageCallback

Function Name	<b>void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
Function Description	Data OUT stage callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd:</b> PCD handle</li><li>• <b>epnum:</b> endpoint number</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 29.2.14 HAL\_PCD\_DataInStageCallback

Function Name	<b>void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
Function Description	Data IN stage callback.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd:</b> PCD handle</li><li>• <b>epnum:</b> endpoint number</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 29.2.15 HAL\_PCD\_SetupStageCallback

Function Name	<b>void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	Setup stage callback.
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>

### 29.2.16 HAL\_PCD\_SOFCallback

Function Name	<b>void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)</b>
---------------	--

Function Description	USB Start Of Frame callback.
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>

### 29.2.17 HAL\_PCD\_ResetCallback

Function Name	<b>void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	USB Reset callback.
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>

### 29.2.18 HAL\_PCD\_SuspendCallback

Function Name	<b>void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	Suspend event callback.
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>

### 29.2.19 HAL\_PCD\_ResumeCallback

Function Name	<b>void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	Resume event callback.
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>

### 29.2.20 HAL\_PCD\_ISOOUTIncompleteCallback

Function Name	<b>void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epi)</b>
Function Description	Incomplete ISO OUT callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>epi:</b> endpoint number</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 29.2.21 HAL\_PCD\_ISOINIncompleteCallback

Function Name	<b>void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epi)</b>
Function Description	Incomplete ISO IN callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd:</b> PCD handle</li> <li>• <b>epi:</b> endpoint number</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 29.2.22 HAL\_PCD\_ConnectCallback

Function Name	<code>void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)</code>
Function Description	Connection event callback.
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>

### 29.2.23 HAL\_PCD\_DisconnectCallback

Function Name	<code>void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)</code>
Function Description	Disconnection event callback.
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>

### 29.2.24 HAL\_PCD\_DevConnect

Function Name	<code>HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)</code>
Function Description	Connect the USB 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>• HAL status</li></ul>

### 29.2.25 HAL\_PCD\_DevDisconnect

Function Name	<code>HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)</code>
Function Description	Disconnect the USB 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>• HAL status</li></ul>

### 29.2.26 HAL\_PCD\_SetAddress

Function Name	<code>HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)</code>
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>• HAL status</li></ul>

### 29.2.27 HAL\_PCD\_EP\_Open

Function Name	<code>HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)</code>
---------------	--

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>• HAL status</li> </ul>

### 29.2.28 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>• HAL status</li> </ul>

### 29.2.29 HAL\_PCD\_EP\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)</b>
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>• HAL status</li> </ul>

### 29.2.30 HAL\_PCD\_EP\_GetRxCount

Function Name	<b>uint16_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)</b>
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>• Data Size</li> </ul>

### 29.2.31 HAL\_PCD\_EP\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t 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>• HAL status</li> </ul>
---------------	--

### 29.2.32 HAL\_PCD\_EP\_SetStall

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t 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>• HAL status</li> </ul>

### 29.2.33 HAL\_PCD\_EP\_ClrStall

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t 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>• HAL status</li> </ul>

### 29.2.34 HAL\_PCD\_EP\_Flush

Function Name	<b>HAL_StatusTypeDef HAL_PCD_EP_Flush (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>• HAL status</li> </ul>

### 29.2.35 HAL\_PCD\_ActivateRemoteWakeup

Function Name	<b>HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)</b>
Function Description	Activate 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>• HAL status</li> </ul>

### 29.2.36 HAL\_PCD\_DeActivateRemoteWakeup

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)</b>
Function Description	De-activate 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>• HAL status</li> </ul>

### 29.2.37 HAL\_PCD\_GetState

Function Name	<b>PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)</b>
Function Description	Return the PCD handle 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>• HAL state</li> </ul>

## 29.3 PCD Firmware driver defines

### 29.3.1 PCD

#### *PCD Exported Macros*

<code>_HAL_PCD_ENABLE</code>	
<code>_HAL_PCD_DISABLE</code>	
<code>_HAL_PCD_GET_FLAG</code>	
<code>_HAL_PCD_CLEAR_FLAG</code>	
<code>_HAL_PCD_IS_INVALID_INTERRUPT</code>	
<code>_HAL_PCD_UNGATE_PHYCLOCK</code>	
<code>_HAL_PCD_GATE_PHYCLOCK</code>	
<code>_HAL_PCD_IS_PHY_SUSPENDED</code>	
<code>USB_OTG_FS_WAKEUP_EXTI_RISING_EDGE</code>	
<code>USB_OTG_FS_WAKEUP_EXTI_FALLING_EDGE</code>	
<code>USB_OTG_FS_WAKEUP_EXTI_RISING_FALLING_EDGE</code>	
<code>USB_OTG_HS_WAKEUP_EXTI_RISING_EDGE</code>	
<code>USB_OTG_HS_WAKEUP_EXTI_FALLING_EDGE</code>	
<code>USB_OTG_HS_WAKEUP_EXTI_RISING_FALLING_EDGE</code>	
<code>USB_OTG_HS_WAKEUP_EXTI_LINE</code>	External interrupt line 20 Connected to the USB HS EXTI Line
<code>USB_OTG_FS_WAKEUP_EXTI_LINE</code>	External interrupt line 18 Connected to the USB FS EXTI Line
<code>_HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_IT</code>	
<code>_HAL_USB_OTG_HS_WAKEUP_EXTI_DISABLE_IT</code>	
<code>_HAL_USB_OTG_HS_WAKEUP_EXTI_GET_FLAG</code>	
<code>_HAL_USB_OTG_HS_WAKEUP_EXTI_CLEAR_FLAG</code>	
<code>_HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_RISING_EDGE</code>	
<code>_HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_FALLING_EDGE</code>	
<code>_HAL_USB_OTG_HS_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE</code>	
<code>_HAL_USB_OTG_HS_WAKEUP_EXTI_GENERATE_SWIT</code>	
<code>_HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_IT</code>	

---

```
_HAL_USB_OTG_FS_WAKEUP_EXTI_DISABLE_IT
_HAL_USB_OTG_FS_WAKEUP_EXTI_GET_FLAG
_HAL_USB_OTG_FS_WAKEUP_EXTI_CLEAR_FLAG
_HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_RISING_EDGE
_HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_FALLING_EDGE
_HAL_USB_OTG_FS_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE
_HAL_USB_OTG_FS_WAKEUP_EXTI_GENERATE_SWIT
```

***PCD Instance definition***

```
IS_PCD_ALL_INSTANCE
```

***PCD PHY Module***

```
PCD_PHY_ULPI
```

```
PCD_PHY_EMBEDDED
```

***PCD Private Macros***

```
PCD_MIN
```

```
PCD_MAX
```

***PCD Speed***

```
PCD_SPEED_HIGH
```

```
PCD_SPEED_HIGH_IN_FULL
```

```
PCD_SPEED_FULL
```

***Turnaround Timeout Value***

```
USBD_HS_TRDT_VALUE
```

```
USBD_FS_TRDT_VALUE
```

## 30 HAL PCD Extension Driver

### 30.1 PCDEEx Firmware driver API description

#### 30.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- [\*HAL\\_PCDEEx\\_SetTxFiFo\(\)\*](#)
- [\*HAL\\_PCDEEx\\_SetRxFiFo\(\)\*](#)

#### 30.1.2 **HAL\_PCDEEx\_SetTxFiFo**

Function Name      **HAL\_StatusTypeDef HAL\_PCDEEx\_SetTxFiFo  
(PCD\_HandleTypeDef \* hpcd, uint8\_t fifo, uint16\_t size)**

Function Description      Set Tx FIFO.

Parameters     

- **hpcd:** PCD handle
- **fifo:** The number of Tx fifo
- **size:** Fifo size

Return values     

- HAL status

#### 30.1.3 **HAL\_PCDEEx\_SetRxFiFo**

Function Name      **HAL\_StatusTypeDef HAL\_PCDEEx\_SetRxFiFo  
(PCD\_HandleTypeDef \* hpcd, uint16\_t size)**

Function Description      Set Rx FIFO.

Parameters     

- **hpcd:** PCD handle
- **size:** Size of Rx fifo

Return values     

- HAL status

## 31 HAL PWR Generic Driver

### 31.1 PWR Firmware driver registers structures

#### 31.1.1 PWR\_PVDTTypeDef

##### Data Fields

- *uint32\_t PVDLevel*
- *uint32\_t Mode*

##### Field Documentation

- *uint32\_t PWR\_PVDTTypeDef::PVDLevel*  
PVDLevel: Specifies the PVD detection level. This parameter can be a value of [PWR\\_PVD\\_detection\\_level](#)
- *uint32\_t PWR\_PVDTTypeDef::Mode*  
Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [PWR\\_PVD\\_Mode](#)

### 31.2 PWR Firmware driver API description

#### 31.2.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses. To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__HAL_RCC_PWR_CLK_ENABLE()` macro.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.

This section contains the following APIs:

- [HAL\\_PWR\\_DeInit\(\)](#)
- [HAL\\_PWR\\_EnableBkUpAccess\(\)](#)
- [HAL\\_PWR\\_DisableBkUpAccess\(\)](#)

#### 31.2.2 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_PWR_PVD_EXTI_ENABLE_IT()` macro.
- The PVD is stopped in Standby mode.

## Wake-up pin configuration

- Wake-up pin is used to wake up the system from Standby mode. This pin is forced in input pull-down configuration and is active on rising edges.
- There is one Wake-up pin: Wake-up Pin 1 on PA.00.

## Low Power modes configuration

The devices feature 3 low-power modes:

- Sleep mode: Cortex-M3 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 STM32F2 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\_PWREx\_EnableFlashPowerDown() function. It can be switched on again by software after exiting the Stop mode using the HAL\_PWREx\_DisableFlashPowerDown() function.

- Entry: The Stop mode is entered using the HAL\_PWR\_EnterSTOPMode(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-M3 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 wake-up, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

### Auto-wake-up (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wake-up event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wake-up mode).
- RTC auto-wake-up (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 Wake-up event, it is necessary to configure the RTC to generate the RTC Wake-up event using the HAL\_RTCEx\_SetWakeUpTimer\_IT() function.

This section contains the following APIs:

- [\*HAL\\_PWR\\_ConfigPVD\(\)\*](#)
- [\*HAL\\_PWR\\_EnablePVD\(\)\*](#)
- [\*HAL\\_PWR\\_DisablePVD\(\)\*](#)
- [\*HAL\\_PWR\\_EnableWakeUpPin\(\)\*](#)
- [\*HAL\\_PWR\\_DisableWakeUpPin\(\)\*](#)
- [\*HAL\\_PWR\\_EnterSLEEPMode\(\)\*](#)
- [\*HAL\\_PWR\\_EnterSTOPMode\(\)\*](#)
- [\*HAL\\_PWR\\_EnterSTANDBYMode\(\)\*](#)
- [\*HAL\\_PWR\\_PVD\\_IRQHandler\(\)\*](#)
- [\*HAL\\_PWR\\_PVDCallback\(\)\*](#)
- [\*HAL\\_PWR\\_EnableSleepOnExit\(\)\*](#)
- [\*HAL\\_PWR\\_DisableSleepOnExit\(\)\*](#)
- [\*HAL\\_PWR\\_EnableSEVOnPend\(\)\*](#)
- [\*HAL\\_PWR\\_DisableSEVOnPend\(\)\*](#)

#### 31.2.3 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.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 31.2.4 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).
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</li> </ul>

---

Backup Domain Access should be kept enabled.

### 31.2.5 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).
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>

### 31.2.6 HAL\_PWR\_ConfigPVD

Function Name	<b>void HAL_PWR_ConfigPVD (PWR_PVDTTypeDef * 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 PWR_PVDTTypeDef 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>

### 31.2.7 HAL\_PWR\_EnablePVD

Function Name	<b>void HAL_PWR_EnablePVD (void )</b>
Function Description	Enables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 31.2.8 HAL\_PWR\_DisablePVD

Function Name	<b>void HAL_PWR_DisablePVD (void )</b>
Function Description	Disables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 31.2.9 HAL\_PWR\_EnableWakeUpPin

Function Name	<b>void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)</b>
Function Description	Enables the Wake-up 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: PWR_WAKEUP_PIN1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 31.2.10 HAL\_PWR\_DisableWakeUpPin

Function Name	<b>void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)</b>
---------------	--

Function Description	Disables the Wake-up 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: PWR_WAKEUP_PIN1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 31.2.11 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: PWR_MAINREGULATOR_ON: SLEEP mode with regulator ONPWR_LOWPOWERREGULATOR_ON: SLEEP mode with low power regulator ON</li> <li><b>SLEEPEntry:</b> Specifies if SLEEP mode is entered with WFI or WFE instruction. This parameter can be one of the following values: PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction</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 STM32F2 family and is kept as parameter just to maintain compatibility with the lower power families.</li> </ul>

### 31.2.12 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: PWR_MAINREGULATOR_ON: Stop mode with regulator ONPWR_LOWPOWERREGULATOR_ON: Stop mode with low power regulator ON</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: PWR_STOPENTRY_WFI: Enter Stop mode with WFI instruction PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction</li> </ul>
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> </ul>

- When exiting Stop mode by issuing an interrupt or a wake-up event, the HSI RC oscillator is selected as system clock.
- When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

### 31.2.13 HAL\_PWR\_EnterSTANDBYMode

Function Name	<b>void HAL_PWR_EnterSTANDBYMode (void )</b>
Function Description	Enters Standby mode.
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>

### 31.2.14 HAL\_PWR\_PVD\_IRQHandler

Function Name	<b>void HAL_PWR_PVD_IRQHandler (void )</b>
Function Description	This function handles the PWR PVD interrupt request.
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>

### 31.2.15 HAL\_PWR\_PVDCallback

Function Name	<b>void HAL_PWR_PVDCallback (void )</b>
Function Description	PWR PVD interrupt callback.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 31.2.16 HAL\_PWR\_EnableSleepOnExit

Function Name	<b>void HAL_PWR_EnableSleepOnExit (void )</b>
Function Description	Indicates Sleep-On-Exit when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.</li> </ul>

### 31.2.17 HAL\_PWR\_DisableSleepOnExit

Function Name	<b>void HAL_PWR_DisableSleepOnExit (void )</b>
Function Description	Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.

---

Return values	• None
Notes	• Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

### 31.2.18 HAL\_PWR\_EnableSEVOnPend

Function Name	<b>void HAL_PWR_EnableSEVOnPend (void )</b>
Function Description	Enables CORTEX M3 SEVONPEND bit.
Return values	• None
Notes	• Sets SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

### 31.2.19 HAL\_PWR\_DisableSEVOnPend

Function Name	<b>void HAL_PWR_DisableSEVOnPend (void )</b>
Function Description	Disables CORTEX M3 SEVONPEND bit.
Return values	• None
Notes	• Clears SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

## 31.3 PWR Firmware driver defines

### 31.3.1 PWR

**PWR CR Register alias address**

DBP\_BIT\_NUMBER  
CR\_DBP\_BB  
PVDE\_BIT\_NUMBER  
CR\_PVDE\_BB

**PWR CSR Register alias address**

EWUP\_BIT\_NUMBER  
CSR\_EWUP\_BB

**PWR Exported Macro**

`_HAL_PWR_GET_FLAG`

**Description:**

- Check PWR flag is set or not.

**Parameters:**

- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - PWR\_FLAG\_WU: Wake Up flag. This flag indicates that a wakeup

event was received from the WKUP pin or from the RTC alarm (Alarm A or Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup. An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.

- PWR\_FLAG\_SB: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
- PWR\_FLAG\_PVDO: PVD Output. This flag is valid only if PVD is enabled by the HAL\_PWR\_EnablePVD() function. The PVD is stopped by Standby mode For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.
- PWR\_FLAG\_BRR: Backup regulator ready flag. This bit is not reset when the device wakes up from Standby mode or by a system reset or power reset.

#### **Return value:**

- The new state of \_\_FLAG\_\_ (TRUE or FALSE).

### [\\_\\_HAL\\_PWR\\_CLEAR\\_FLAG](#)

#### **Description:**

- Clear the PWR's pending flags.

#### **Parameters:**

- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be one of the following values:
  - PWR\_FLAG\_WU: Wake Up flag
  - PWR\_FLAG\_SB: StandBy flag

### [\\_\\_HAL\\_PWR\\_PVD\\_EXTI\\_ENABLE\\_IT](#)

#### **Description:**

- Enable the PVD EXTI Line 16.

#### **Return value:**

- None.

### [\\_\\_HAL\\_PWR\\_PVD\\_EXTI\\_DISABLE\\_IT](#)

#### **Description:**

- Disable the PVD EXTI Line 16.

#### **Return value:**

- None.

### [\\_\\_HAL\\_PWR\\_PVD\\_EXTI\\_ENABLE\\_EVENT\\_T](#)

#### **Description:**

- Enable event on PVD EXTI Line 16.

`__HAL_PWR_PVD_EXTI_DISABLE_EVENT`

**Return value:**

- None.

**Description:**

- Disable event on PVD Exti Line 16.

`__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE`

**Return value:**

- None.

**Description:**

- Enable the PVD Extended Interrupt Rising Trigger.

`__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE`

**Return value:**

- None.

**Description:**

- Disable the PVD Extended Interrupt Rising Trigger.

`__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE`

**Return value:**

- None.

**Description:**

- Enable the PVD Extended Interrupt Falling Trigger.

`__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE`

**Return value:**

- None.

**Description:**

- Disable the PVD Extended Interrupt Falling Trigger.

`__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE`

**Return value:**

- None.

**Description:**

- PVD EXTI line configuration: set rising & falling edge trigger.

`__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE`

**Return value:**

- None.

**Description:**

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

`__HAL_PWR_PVD_EXTI_GET_FLAG`

**Return value:**

- None.

**Description:**

- checks whether the specified PVD Exti

---

interrupt flag is set or not.

**Return value:**

- EXTI: PVD Line Status.

`__HAL_PWR_PVD_EXTI_CLEAR_FLAG`

**Description:**

- Clear the PVD Exti flag.

**Return value:**

- None.

`__HAL_PWR_PVD_EXTI_GENERATE_SWIT`

**Description:**

- Generates a Software interrupt on PVD EXTI line.

**Return value:**

- None

**PWR Flag**

`PWR_FLAG_WU`

`PWR_FLAG_SB`

`PWR_FLAG_PVDO`

`PWR_FLAG_BRR`

**PWR Private macros to check input parameters**

`IS_PWR_WAKEUP_PIN`

`IS_PWR_PVD_LEVEL`

`IS_PWR_PVD_MODE`

`IS_PWR_REGULATOR`

`IS_PWR_SLEEP_ENTRY`

`IS_PWR_STOP_ENTRY`

**PWR PVD detection level**

`PWR_PVDLEVEL_0`

`PWR_PVDLEVEL_1`

`PWR_PVDLEVEL_2`

`PWR_PVDLEVEL_3`

`PWR_PVDLEVEL_4`

`PWR_PVDLEVEL_5`

`PWR_PVDLEVEL_6`

`PWR_PVDLEVEL_7`

**PWR PVD EXTI Line**

`PWR_EXTI_LINE_PVD` External interrupt line 16 Connected to the PVD EXTI Line

**PWR PVD Mode**

PWR_PVD_MODE_NORMAL	basic mode is used
PWR_PVD_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
PWR_PVD_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
PWR_PVD_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING	Event Mode with Rising edge trigger detection
PWR_PVD_MODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection

**PWR PVD Mode Mask**

PVD\_MODE\_IT

PVD\_MODE\_EVT

PVD\_RISING\_EDGE

PVD\_FALLING\_EDGE

**PWR Register alias address**

PWR\_OFFSET

PWR\_CR\_OFFSET

PWR\_CSR\_OFFSET

PWR\_CR\_OFFSET\_BB

PWR\_CSR\_OFFSET\_BB

**PWR Regulator state in SLEEP/STOP mode**

PWR\_MAINREGULATOR\_ON

PWR\_LOWPOWERREGULATOR\_ON

**PWR SLEEP mode entry**

PWR\_SLEEPENTRY\_WFI

PWR\_SLEEPENTRY\_WFE

**PWR STOP mode entry**

PWR\_STOPENTRY\_WFI

PWR\_STOPENTRY\_WFE

**PWR WakeUp Pins**

PWR\_WAKEUP\_PIN1

## 32 HAL PWR Extension Driver

### 32.1 PWREx Firmware driver API description

#### 32.1.1 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\_PWREx\_EnableBkUpReg() function to enable the low power backup regulator.
- When the backup domain is supplied by VDD (analog switch connected to VDD) the backup SRAM is powered from VDD which replaces the VBAT power supply to save battery life.
- The backup SRAM is not mass erased by a tamper event. It is read protected to prevent confidential data, such as cryptographic private key, from being accessed. The backup SRAM can be erased only through the Flash interface when a protection level change from level 1 to level 0 is requested. Refer to the description of Read protection (RDP) in the Flash programming manual. 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\_PWREx\_EnableFlashPowerDown() function, the Flash memory also enters power down mode when the device enters Stop mode. When the Flash memory is in power down mode, an additional startup delay is incurred when waking up from Stop mode.

This section contains the following APIs:

- [\*HAL\\_PWREx\\_EnableBkUpReg\(\)\*](#)
- [\*HAL\\_PWREx\\_DisableBkUpReg\(\)\*](#)
- [\*HAL\\_PWREx\\_EnableFlashPowerDown\(\)\*](#)
- [\*HAL\\_PWREx\\_DisableFlashPowerDown\(\)\*](#)

#### 32.1.2 HAL\_PWREx\_EnableBkUpReg

Function Name      **HAL\_StatusTypeDef HAL\_PWREx\_EnableBkUpReg (void )**

Function Description      Enables the Backup Regulator.

Return values     

- HAL status

#### 32.1.3 HAL\_PWREx\_DisableBkUpReg

Function Name      **HAL\_StatusTypeDef HAL\_PWREx\_DisableBkUpReg (void )**

Function Description      Disables the Backup Regulator.

Return values • HAL status

### 32.1.4 HAL\_PWREx\_EnableFlashPowerDown

Function Name **void HAL\_PWREx\_EnableFlashPowerDown (void )**

Function Description Enables the Flash Power Down in Stop mode.

Return values • None

### 32.1.5 HAL\_PWREx\_DisableFlashPowerDown

Function Name **void HAL\_PWREx\_DisableFlashPowerDown (void )**

Function Description Disables the Flash Power Down in Stop mode.

Return values • None

## 32.2 PWREx Firmware driver defines

### 32.2.1 PWREx

*PWRx CSR Register alias address*

BRE\_BIT\_NUMBER

CSR\_BRE\_BB

*PWREx Private Constants*

PWR\_BKPREG\_TIMEOUT\_VALUE

*PWREx Register alias address*

FPDS\_BIT\_NUMBER

CR\_FPDS\_BB

## 33 HAL RCC Generic Driver

### 33.1 RCC Firmware driver registers structures

#### 33.1.1 RCC\_PLLInitTypeDef

##### 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

#### 33.1.2 RCC\_OscInitTypeDef

##### 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::HSIState***  
The new state of the HSI. This parameter can be a value of [\*\*RCC\\_HSI\\_Config\*\*](#)
- ***uint32\_t RCC\_OscInitTypeDef::HSICalibrationValue***  
The calibration trimming value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F
- ***uint32\_t RCC\_OscInitTypeDef::LSIState***  
The new state of the LSI. This parameter can be a value of [\*\*RCC\\_LSI\\_Config\*\*](#)
- ***RCC\_PLLInitTypeDef RCC\_OscInitTypeDef::PLL***  
PLL structure parameters

**33.1.3 RCC\_ClkInitTypeDef****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\*\*](#)

## 33.2 RCC Firmware driver API description

### 33.2.1 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)

### 33.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Possible Workarounds:

1. Enable the peripheral clock sometimes before the peripheral read/write register is required.
2. For AHB peripheral, insert two dummy read to the peripheral register.
3. For APB peripheral, insert a dummy read to the peripheral register.

### 33.2.3 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 120 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 clocks automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.
  7. MCO1 (microcontroller clock output), used to output 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. 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 stm32f2xx devices, the maximum frequency of the SYSCLK and HCLK is 120 MHz, PCLK2 60 MHz and PCLK1 30 MHz. Depending on the device voltage range, the maximum frequency should be adapted according to the table below.

**Table 16: Number of wait states (WS) according to CPU clock (HCLK) frequency**

Latency	HCLK clock frequency (MHz)			
	voltage range 2.7 to 3.6V	voltage range 2.4 to 2.7V	voltage range 2.1 to 2.4V	voltage range 1.8 to 2.1V
0W (1 CPU cycle)	0 < HCLK ≤ 30	0 < HCLK ≤ 24	0 < HCLK ≤ 18	0 < HCLK ≤ 16
1WS (2 CPU cycles)	30 < HCLK ≤ 60	24 < HCLK ≤ 48	18 < HCLK ≤ 36	16 < HCLK ≤ 32
2WS (3 CPU cycles)	60 < HCLK ≤ 90	48 < HCLK ≤ 72	36 < HCLK ≤ 54	32 < HCLK ≤ 48
3WS (4 CPU cycles)	90 < HCLK ≤ 120	72 < HCLK ≤ 96	54 < HCLK ≤ 72	48 < HCLK ≤ 64
4WS (5 CPU cycles)	NA	96 < HCLK ≤ 120	72 < HCLK ≤ 90	64 < HCLK ≤ 80
5WS (6 CPU cycles)	NA	NA	90 < HCLK ≤ 108	80 < HCLK ≤ 96
6WS (7 CPU cycles)	NA	NA	108 < HCLK ≤ 120	96 < HCLK ≤ 112

Latency	HCLK clock frequency (MHz)			
	voltage range 2.7 to 3.6V	voltage range 2.4 to 2.7V	voltage range 2.1 to 2.4V	voltage range 1.8 to 2.1V
7WS (8 CPU cycles)	NA	NA	NA	112 < HCLK ≤ 120

This section contains the following APIs:

- [\*HAL\\_RCC\\_DelInit\(\)\*](#)
- [\*HAL\\_RCC\\_OscConfig\(\)\*](#)
- [\*HAL\\_RCC\\_ClockConfig\(\)\*](#)

### 33.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- [\*HAL\\_RCC\\_MCOConfig\(\)\*](#)
- [\*HAL\\_RCC\\_EnableCSS\(\)\*](#)
- [\*HAL\\_RCC\\_DisableCSS\(\)\*](#)
- [\*HAL\\_RCC\\_GetSysClockFreq\(\)\*](#)
- [\*HAL\\_RCC\\_GetHCLKFreq\(\)\*](#)
- [\*HAL\\_RCC\\_GetPCLK1Freq\(\)\*](#)
- [\*HAL\\_RCC\\_GetPCLK2Freq\(\)\*](#)
- [\*HAL\\_RCC\\_GetOscConfig\(\)\*](#)
- [\*HAL\\_RCC\\_GetClockConfig\(\)\*](#)
- [\*HAL\\_RCC\\_NMI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_RCC\\_CSSCallback\(\)\*](#)

### 33.2.5 HAL\_RCC\_DelInit

Function Name	<b>void HAL_RCC_DelInit (void )</b>
Function Description	Resets the RCC clock configuration to the default reset state.
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>

### 33.2.6 HAL\_RCC\_OscConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)</b>
Function Description	Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_OscInitStruct:</b> pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.</li> </ul>

- |               |  |
|---------------|--|
| Return values | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |
| Notes         | <ul style="list-style-type: none"> <li>• The PLL is not disabled when used as system clock.</li> </ul> |

### 33.2.7 HAL\_RCC\_ClockConfig

Function Name	<b>HAL_StatusTypeDef HAL_RCC_ClockConfig (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 RCC_ClkInitStruct.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_ClkInitStruct:</b> pointer to an RCC_OsclInitTypeDef 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 HAL_RCC_GetHCLKFreq() 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>

### 33.2.8 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: RCC_MCO1: Clock source to output on MCO1 pin(PA8).RCC_MCO2: Clock source to output on MCO2 pin(PC9).</li> <li>• <b>RCC_MCOsource:</b> specifies the clock source to output. This parameter can be one of the following values: RCC_MCO1SOURCE_HSI: HSI clock selected as MCO1 sourceRCC_MCO1SOURCE_LSE: LSE clock selected as MCO1 sourceRCC_MCO1SOURCE_HSE: HSE clock selected as MCO1 sourceRCC_MCO1SOURCE_PLLCLK: main PLL clock selected as MCO1</li> </ul>

sourceRCC\_MCO2SOURCE\_SYSCLK: System clock (SYSCLK) selected as MCO2  
 sourceRCC\_MCO2SOURCE\_PLLI2SCLK: PLLI2S clock selected as MCO2 sourceRCC\_MCO2SOURCE\_HSE: HSE clock selected as MCO2  
 sourceRCC\_MCO2SOURCE\_PLLCLK: main PLL clock selected as MCO2 source

- **RCC\_MCODiv:** specifies the MCOx prescaler. This parameter can be one of the following values:  
 RCC\_MCODIV\_1: no division applied to MCOx  
 clockRCC\_MCODIV\_2: division by 2 applied to MCOx  
 clockRCC\_MCODIV\_3: division by 3 applied to MCOx  
 clockRCC\_MCODIV\_4: division by 4 applied to MCOx  
 clockRCC\_MCODIV\_5: division by 5 applied to MCOx clock

Return values

- None

Notes

- PA8/PC9 should be configured in alternate function mode.

### 33.2.9 HAL\_RCC\_EnableCSS

Function Name **void HAL\_RCC\_EnableCSS (void )**

Function Description Enables the Clock Security System.

Return values

- None

Notes

- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.

### 33.2.10 HAL\_RCC\_DisableCSS

Function Name **void HAL\_RCC\_DisableCSS (void )**

Function Description Disables the Clock Security System.

Return values

- None

### 33.2.11 HAL\_RCC\_GetSysClockFreq

Function Name **uint32\_t HAL\_RCC\_GetSysClockFreq (void )**

Function Description Returns the SYSCLK frequency.

Return values

- SYSCLK frequency

Notes

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is HSI, function returns values based on HSI\_VALUE(\*)
- If SYSCLK source is HSE, function returns values based on HSE\_VALUE(\*\*)
- If SYSCLK source is PLL, function returns values based on HSE\_VALUE(\*\*) or HSI\_VALUE(\*) multiplied/divided by the

PLL factors.

- (\*) HSI\_VALUE is a constant defined in `stm32f2xx_hal_conf.h` file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (\*\*) HSE\_VALUE is a constant defined in `stm32f2xx_hal_conf.h` file (default value 25 MHz), user has to ensure that HSE\_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

### 33.2.12 HAL\_RCC\_GetHCLKFreq

Function Name	<code>uint32_t HAL_RCC_GetHCLKFreq (void )</code>
Function Description	Returns the HCLK frequency.
Return values	<ul style="list-style-type: none"> <li>• HCLK frequency</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>

### 33.2.13 HAL\_RCC\_GetPCLK1Freq

Function Name	<code>uint32_t HAL_RCC_GetPCLK1Freq (void )</code>
Function Description	Returns the PCLK1 frequency.
Return values	<ul style="list-style-type: none"> <li>• PCLK1 frequency</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>

### 33.2.14 HAL\_RCC\_GetPCLK2Freq

Function Name	<code>uint32_t HAL_RCC_GetPCLK2Freq (void )</code>
Function Description	Returns the PCLK2 frequency.
Return values	<ul style="list-style-type: none"> <li>• PCLK2 frequency</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>

### 33.2.15 HAL\_RCC\_GetOscConfig

Function Name	<code>void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * </code>
---------------	---

**RCC\_OsclInitStruct)**

Function Description	Configures the RCC_OsclInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_OsclInitStruct:</b> pointer to an RCC_OsclInitTypeDef structure that will be configured.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**33.2.16 HAL\_RCC\_GetClockConfig**

Function Name	<b>void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)</b>
Function Description	Configures the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>RCC_ClkInitStruct:</b> pointer to an RCC_ClkInitTypeDef 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>

**33.2.17 HAL\_RCC\_NMI\_IRQHandler**

Function Name	<b>void HAL_RCC_NMI_IRQHandler (void )</b>
Function Description	This function handles the RCC CSS interrupt request.
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>

**33.2.18 HAL\_RCC\_CSSCallback**

Function Name	<b>void HAL_RCC_CSSCallback (void )</b>
Function Description	RCC Clock Security System interrupt callback.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**33.3 RCC Firmware driver defines****33.3.1 RCC*****AHB1 Peripheral Clock Enable Disable***

```

__HAL_RCC_GPIOA_CLK_ENABLE
__HAL_RCC_GPIOB_CLK_ENABLE
__HAL_RCC_GPIOC_CLK_ENABLE
__HAL_RCC_GPIOD_CLK_ENABLE
__HAL_RCC_GPIOE_CLK_ENABLE
__HAL_RCC_GPIOF_CLK_ENABLE
__HAL_RCC_GPIOG_CLK_ENABLE
__HAL_RCC_GPIOH_CLK_ENABLE

```

```
_HAL_RCC_GPIOI_CLK_ENABLE  
_HAL_RCC_CRC_CLK_ENABLE  
_HAL_RCC_BKPSRAM_CLK_ENABLE  
_HAL_RCC_DMA1_CLK_ENABLE  
_HAL_RCC_DMA2_CLK_ENABLE  
_HAL_RCC_USB_OTG_HS_CLK_ENABLE  
_HAL_RCC_USB_OTG_HS_ULPI_CLK_ENABLE  
_HAL_RCC_GPIOA_CLK_DISABLE  
_HAL_RCC_GPIOB_CLK_DISABLE  
_HAL_RCC_GPIOC_CLK_DISABLE  
_HAL_RCC_GPIOD_CLK_DISABLE  
_HAL_RCC_GPIOE_CLK_DISABLE  
_HAL_RCC_GPIOF_CLK_DISABLE  
_HAL_RCC_GPIOG_CLK_DISABLE  
_HAL_RCC_GPIOH_CLK_DISABLE  
_HAL_RCC_GPIOI_CLK_DISABLE  
_HAL_RCC_CRC_CLK_DISABLE  
_HAL_RCC_BKPSRAM_CLK_DISABLE  
_HAL_RCC_DMA1_CLK_DISABLE  
_HAL_RCC_DMA2_CLK_DISABLE  
_HAL_RCC_USB_OTG_HS_CLK_DISABLE  
_HAL_RCC_USB_OTG_HS_ULPI_CLK_DISABLE
```

**AHB1 Force Release Reset**

```
_HAL_RCC_AHB1_FORCE_RESET  
_HAL_RCC_GPIOA_FORCE_RESET  
_HAL_RCC_GPIOB_FORCE_RESET  
_HAL_RCC_GPIOC_FORCE_RESET  
_HAL_RCC_GPIOD_FORCE_RESET  
_HAL_RCC_GPIOE_FORCE_RESET  
_HAL_RCC_GPIOF_FORCE_RESET  
_HAL_RCC_GPIOG_FORCE_RESET  
_HAL_RCC_GPIOH_FORCE_RESET  
_HAL_RCC_GPIOI_FORCE_RESET  
_HAL_RCC_CRC_FORCE_RESET  
_HAL_RCC_DMA1_FORCE_RESET  
_HAL_RCC_DMA2_FORCE_RESET
```

```
_HAL_RCC_USB_OTG_HS_FORCE_RESET  
_HAL_RCC_OTGHSULPI_FORCE_RESET  
_HAL_RCC_AHB1_RELEASE_RESET  
_HAL_RCC_GPIOA_RELEASE_RESET  
_HAL_RCC_GPIOB_RELEASE_RESET  
_HAL_RCC_GPIOC_RELEASE_RESET  
_HAL_RCC_GPIOD_RELEASE_RESET  
_HAL_RCC_GPIOE_RELEASE_RESET  
_HAL_RCC_GPIOF_RELEASE_RESET  
_HAL_RCC_GPIOG_RELEASE_RESET  
_HAL_RCC_GPIOH_RELEASE_RESET  
_HAL_RCC_GPIOI_RELEASE_RESET  
_HAL_RCC_CRC_RELEASE_RESET  
_HAL_RCC_DMA1_RELEASE_RESET  
_HAL_RCC_DMA2_RELEASE_RESET  
_HAL_RCC_USB_OTG_HS_RELEASE_RESET  
_HAL_RCC_OTGHSULPI_RELEASE_RESET
```

**AHB1 Peripheral Low Power Enable Disable**

```
_HAL_RCC_GPIOA_CLK_SLEEP_ENABLE  
_HAL_RCC_GPIOB_CLK_SLEEP_ENABLE  
_HAL_RCC_GPIOC_CLK_SLEEP_ENABLE  
_HAL_RCC_GPIOD_CLK_SLEEP_ENABLE  
_HAL_RCC_GPIOE_CLK_SLEEP_ENABLE  
_HAL_RCC_GPIOF_CLK_SLEEP_ENABLE  
_HAL_RCC_GPIOG_CLK_SLEEP_ENABLE  
_HAL_RCC_GPIOH_CLK_SLEEP_ENABLE  
_HAL_RCC_GPIOI_CLK_SLEEP_ENABLE  
_HAL_RCC_CRC_CLK_SLEEP_ENABLE  
_HAL_RCC_FLITF_CLK_SLEEP_ENABLE  
_HAL_RCC_SRAM1_CLK_SLEEP_ENABLE  
_HAL_RCC_SRAM2_CLK_SLEEP_ENABLE  
_HAL_RCC_BKPSRAM_CLK_SLEEP_ENABLE  
_HAL_RCC_DMA1_CLK_SLEEP_ENABLE  
_HAL_RCC_DMA2_CLK_SLEEP_ENABLE  
_HAL_RCC_USB_OTG_HS_CLK_SLEEP_ENABLE  
_HAL_RCC_USB_OTG_HS_ULPI_CLK_SLEEP_ENABLE
```

```
_HAL_RCC_GPIOA_CLK_SLEEP_DISABLE  
_HAL_RCC_GPIOB_CLK_SLEEP_DISABLE  
_HAL_RCC_GPIOC_CLK_SLEEP_DISABLE  
_HAL_RCC_GPIOD_CLK_SLEEP_DISABLE  
_HAL_RCC_GPIOE_CLK_SLEEP_DISABLE  
_HAL_RCC_GPIOF_CLK_SLEEP_DISABLE  
_HAL_RCC_GPIOG_CLK_SLEEP_DISABLE  
_HAL_RCC_GPIOH_CLK_SLEEP_DISABLE  
_HAL_RCC_GPIOI_CLK_SLEEP_DISABLE  
_HAL_RCC_CRC_CLK_SLEEP_DISABLE  
_HAL_RCC_FLITF_CLK_SLEEP_DISABLE  
_HAL_RCC_SRAM1_CLK_SLEEP_DISABLE  
_HAL_RCC_SRAM2_CLK_SLEEP_DISABLE  
_HAL_RCC_BKPSRAM_CLK_SLEEP_DISABLE  
_HAL_RCC_DMA1_CLK_SLEEP_DISABLE  
_HAL_RCC_DMA2_CLK_SLEEP_DISABLE  
_HAL_RCC_USB_OTG_HS_CLK_SLEEP_DISABLE  
_HAL_RCC_USB_OTG_HS_ULPI_CLK_SLEEP_DISABLE
```

**AHB2 Peripheral Clock Enable Disable**

```
_HAL_RCC_USB_OTG_FS_CLK_ENABLE  
_HAL_RCC_RNG_CLK_ENABLE  
_HAL_RCC_USB_OTG_FS_CLK_DISABLE  
_HAL_RCC_RNG_CLK_DISABLE
```

**AHB2 Force Release Reset**

```
_HAL_RCC_AHB2_FORCE_RESET  
_HAL_RCC_RNG_FORCE_RESET  
_HAL_RCC_USB_OTG_FS_FORCE_RESET  
_HAL_RCC_AHB2_RELEASE_RESET  
_HAL_RCC_RNG_RELEASE_RESET  
_HAL_RCC_USB_OTG_FS_RELEASE_RESET
```

**AHB2 Peripheral Low Power Enable Disable**

```
_HAL_RCC_USB_OTG_FS_CLK_SLEEP_ENABLE  
_HAL_RCC_RNG_CLK_SLEEP_ENABLE  
_HAL_RCC_USB_OTG_FS_CLK_SLEEP_DISABLE  
_HAL_RCC_RNG_CLK_SLEEP_DISABLE
```

**AHB3 Peripheral Clock Enable Disable**

`_HAL_RCC_FSMC_CLK_ENABLE`  
`_HAL_RCC_FSMC_CLK_DISABLE`

**AHB3 Force Release Reset**

`_HAL_RCC_AHB3_FORCE_RESET`  
`_HAL_RCC_FSMC_FORCE_RESET`  
`_HAL_RCC_AHB3_RELEASE_RESET`  
`_HAL_RCC_FSMC_RELEASE_RESET`

**AHB Clock Source**

`RCC_SYSCLK_DIV1`  
`RCC_SYSCLK_DIV2`  
`RCC_SYSCLK_DIV4`  
`RCC_SYSCLK_DIV8`  
`RCC_SYSCLK_DIV16`  
`RCC_SYSCLK_DIV64`  
`RCC_SYSCLK_DIV128`  
`RCC_SYSCLK_DIV256`  
`RCC_SYSCLK_DIV512`

**APB1/APB2 Clock Source**

`RCC_HCLK_DIV1`  
`RCC_HCLK_DIV2`  
`RCC_HCLK_DIV4`  
`RCC_HCLK_DIV8`  
`RCC_HCLK_DIV16`

**APB1 Peripheral Clock Enable Disable**

`_HAL_RCC_TIM2_CLK_ENABLE`  
`_HAL_RCC_TIM3_CLK_ENABLE`  
`_HAL_RCC_TIM4_CLK_ENABLE`  
`_HAL_RCC_TIM5_CLK_ENABLE`  
`_HAL_RCC_TIM6_CLK_ENABLE`  
`_HAL_RCC_TIM7_CLK_ENABLE`  
`_HAL_RCC_TIM12_CLK_ENABLE`  
`_HAL_RCC_TIM13_CLK_ENABLE`  
`_HAL_RCC_TIM14_CLK_ENABLE`  
`_HAL_RCC_WWDG_CLK_ENABLE`  
`_HAL_RCC_SPI2_CLK_ENABLE`  
`_HAL_RCC_SPI3_CLK_ENABLE`

```
_HAL_RCC_USART2_CLK_ENABLE  
_HAL_RCC_USART3_CLK_ENABLE  
_HAL_RCC_UART4_CLK_ENABLE  
_HAL_RCC_UART5_CLK_ENABLE  
_HAL_RCC_I2C1_CLK_ENABLE  
_HAL_RCC_I2C2_CLK_ENABLE  
_HAL_RCC_I2C3_CLK_ENABLE  
_HAL_RCC_CAN1_CLK_ENABLE  
_HAL_RCC_CAN2_CLK_ENABLE  
_HAL_RCC_PWR_CLK_ENABLE  
_HAL_RCC_DAC_CLK_ENABLE  
_HAL_RCC_TIM2_CLK_DISABLE  
_HAL_RCC_TIM3_CLK_DISABLE  
_HAL_RCC_TIM4_CLK_DISABLE  
_HAL_RCC_TIM5_CLK_DISABLE  
_HAL_RCC_TIM6_CLK_DISABLE  
_HAL_RCC_TIM7_CLK_DISABLE  
_HAL_RCC_TIM12_CLK_DISABLE  
_HAL_RCC_TIM13_CLK_DISABLE  
_HAL_RCC_TIM14_CLK_DISABLE  
_HAL_RCC_WWDG_CLK_DISABLE  
_HAL_RCC_SPI2_CLK_DISABLE  
_HAL_RCC_SPI3_CLK_DISABLE  
_HAL_RCC_USART2_CLK_DISABLE  
_HAL_RCC_USART3_CLK_DISABLE  
_HAL_RCC_UART4_CLK_DISABLE  
_HAL_RCC_UART5_CLK_DISABLE  
_HAL_RCC_I2C1_CLK_DISABLE  
_HAL_RCC_I2C2_CLK_DISABLE  
_HAL_RCC_I2C3_CLK_DISABLE  
_HAL_RCC_PWR_CLK_DISABLE  
_HAL_RCC_CAN1_CLK_DISABLE  
_HAL_RCC_CAN2_CLK_DISABLE  
_HAL_RCC_DAC_CLK_DISABLE  
APB1 Force Release Reset  
_HAL_RCC_APB1_FORCE_RESET
```

```
_HAL_RCC_TIM2_FORCE_RESET  
_HAL_RCC_TIM3_FORCE_RESET  
_HAL_RCC_TIM4_FORCE_RESET  
_HAL_RCC_TIM5_FORCE_RESET  
_HAL_RCC_TIM6_FORCE_RESET  
_HAL_RCC_TIM7_FORCE_RESET  
_HAL_RCC_TIM12_FORCE_RESET  
_HAL_RCC_TIM13_FORCE_RESET  
_HAL_RCC_TIM14_FORCE_RESET  
_HAL_RCC_WWDG_FORCE_RESET  
_HAL_RCC_SPI2_FORCE_RESET  
_HAL_RCC_SPI3_FORCE_RESET  
_HAL_RCC_USART2_FORCE_RESET  
_HAL_RCC_USART3_FORCE_RESET  
_HAL_RCC_UART4_FORCE_RESET  
_HAL_RCC_UART5_FORCE_RESET  
_HAL_RCC_I2C1_FORCE_RESET  
_HAL_RCC_I2C2_FORCE_RESET  
_HAL_RCC_I2C3_FORCE_RESET  
_HAL_RCC_CAN1_FORCE_RESET  
_HAL_RCC_CAN2_FORCE_RESET  
_HAL_RCC_PWR_FORCE_RESET  
_HAL_RCC_DAC_FORCE_RESET  
_HAL_RCC_APB1_RELEASE_RESET  
_HAL_RCC_TIM2_RELEASE_RESET  
_HAL_RCC_TIM3_RELEASE_RESET  
_HAL_RCC_TIM4_RELEASE_RESET  
_HAL_RCC_TIM5_RELEASE_RESET  
_HAL_RCC_TIM6_RELEASE_RESET  
_HAL_RCC_TIM7_RELEASE_RESET  
_HAL_RCC_TIM12_RELEASE_RESET  
_HAL_RCC_TIM13_RELEASE_RESET  
_HAL_RCC_TIM14_RELEASE_RESET  
_HAL_RCC_WWDG_RELEASE_RESET  
_HAL_RCC_SPI2_RELEASE_RESET  
_HAL_RCC_SPI3_RELEASE_RESET
```

```
_HAL_RCC_USART2_RELEASE_RESET  
_HAL_RCC_USART3_RELEASE_RESET  
_HAL_RCC_UART4_RELEASE_RESET  
_HAL_RCC_UART5_RELEASE_RESET  
_HAL_RCC_I2C1_RELEASE_RESET  
_HAL_RCC_I2C2_RELEASE_RESET  
_HAL_RCC_I2C3_RELEASE_RESET  
_HAL_RCC_CAN1_RELEASE_RESET  
_HAL_RCC_CAN2_RELEASE_RESET  
_HAL_RCC_PWR_RELEASE_RESET  
_HAL_RCC_DAC_RELEASE_RESET
```

***APB1 Peripheral Low Power Enable Disable***

```
_HAL_RCC_TIM2_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM3_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM4_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM5_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM6_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM7_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM12_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM13_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM14_CLK_SLEEP_ENABLE  
_HAL_RCC_WWDG_CLK_SLEEP_ENABLE  
_HAL_RCC_SPI2_CLK_SLEEP_ENABLE  
_HAL_RCC_SPI3_CLK_SLEEP_ENABLE  
_HAL_RCC_USART2_CLK_SLEEP_ENABLE  
_HAL_RCC_USART3_CLK_SLEEP_ENABLE  
_HAL_RCC_UART4_CLK_SLEEP_ENABLE  
_HAL_RCC_UART5_CLK_SLEEP_ENABLE  
_HAL_RCC_I2C1_CLK_SLEEP_ENABLE  
_HAL_RCC_I2C2_CLK_SLEEP_ENABLE  
_HAL_RCC_I2C3_CLK_SLEEP_ENABLE  
_HAL_RCC_PWR_CLK_SLEEP_ENABLE  
_HAL_RCC_CAN1_CLK_SLEEP_ENABLE  
_HAL_RCC_CAN2_CLK_SLEEP_ENABLE  
_HAL_RCC_DAC_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM2_CLK_SLEEP_DISABLE
```

```
_HAL_RCC_TIM3_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM4_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM5_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM6_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM7_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM12_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM13_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM14_CLK_SLEEP_DISABLE  
_HAL_RCC_WWDG_CLK_SLEEP_DISABLE  
_HAL_RCC_SPI2_CLK_SLEEP_DISABLE  
_HAL_RCC_SPI3_CLK_SLEEP_DISABLE  
_HAL_RCC_USART2_CLK_SLEEP_DISABLE  
_HAL_RCC_USART3_CLK_SLEEP_DISABLE  
_HAL_RCC_UART4_CLK_SLEEP_DISABLE  
_HAL_RCC_UART5_CLK_SLEEP_DISABLE  
_HAL_RCC_I2C1_CLK_SLEEP_DISABLE  
_HAL_RCC_I2C2_CLK_SLEEP_DISABLE  
_HAL_RCC_I2C3_CLK_SLEEP_DISABLE  
_HAL_RCC_PWR_CLK_SLEEP_DISABLE  
_HAL_RCC_CAN1_CLK_SLEEP_DISABLE  
_HAL_RCC_CAN2_CLK_SLEEP_DISABLE  
_HAL_RCC_DAC_CLK_SLEEP_DISABLE
```

**APB2 Peripheral Clock Enable Disable**

```
_HAL_RCC_TIM1_CLK_ENABLE  
_HAL_RCC_TIM8_CLK_ENABLE  
_HAL_RCC_USART1_CLK_ENABLE  
_HAL_RCC_USART6_CLK_ENABLE  
_HAL_RCC_ADC1_CLK_ENABLE  
_HAL_RCC_ADC2_CLK_ENABLE  
_HAL_RCC_ADC3_CLK_ENABLE  
_HAL_RCC_SDIO_CLK_ENABLE  
_HAL_RCC_SPI1_CLK_ENABLE  
_HAL_RCC_SYSCFG_CLK_ENABLE  
_HAL_RCC_TIM9_CLK_ENABLE  
_HAL_RCC_TIM10_CLK_ENABLE  
_HAL_RCC_TIM11_CLK_ENABLE
```

```
_HAL_RCC_TIM1_CLK_DISABLE  
_HAL_RCC_TIM8_CLK_DISABLE  
_HAL_RCC_USART1_CLK_DISABLE  
_HAL_RCC_USART6_CLK_DISABLE  
_HAL_RCC_ADC1_CLK_DISABLE  
_HAL_RCC_ADC2_CLK_DISABLE  
_HAL_RCC_ADC3_CLK_DISABLE  
_HAL_RCC_SDIO_CLK_DISABLE  
_HAL_RCC_SPI1_CLK_DISABLE  
_HAL_RCC_SYSCFG_CLK_DISABLE  
_HAL_RCC_TIM9_CLK_DISABLE  
_HAL_RCC_TIM10_CLK_DISABLE  
_HAL_RCC_TIM11_CLK_DISABLE  
APB2 Force Release Reset  
_HAL_RCC_APB2_FORCE_RESET  
_HAL_RCC_TIM1_FORCE_RESET  
_HAL_RCC_TIM8_FORCE_RESET  
_HAL_RCC_USART1_FORCE_RESET  
_HAL_RCC_USART6_FORCE_RESET  
_HAL_RCC_ADC_FORCE_RESET  
_HAL_RCC_SDIO_FORCE_RESET  
_HAL_RCC_SPI1_FORCE_RESET  
_HAL_RCC_SYSCFG_FORCE_RESET  
_HAL_RCC_TIM9_FORCE_RESET  
_HAL_RCC_TIM10_FORCE_RESET  
_HAL_RCC_TIM11_FORCE_RESET  
_HAL_RCC_APB2_RELEASE_RESET  
_HAL_RCC_TIM1_RELEASE_RESET  
_HAL_RCC_TIM8_RELEASE_RESET  
_HAL_RCC_USART1_RELEASE_RESET  
_HAL_RCC_USART6_RELEASE_RESET  
_HAL_RCC_ADC_RELEASE_RESET  
_HAL_RCC_SDIO_RELEASE_RESET  
_HAL_RCC_SPI1_RELEASE_RESET  
_HAL_RCC_SYSCFG_RELEASE_RESET  
_HAL_RCC_TIM9_RELEASE_RESET
```

```
_HAL_RCC_TIM10_RELEASE_RESET  
_HAL_RCC_TIM11_RELEASE_RESET  
APB2 Peripheral Low Power Enable Disable  
_HAL_RCC_TIM1_CLK_SLEEP_ENABLE  
_HAL_RCC_USART1_CLK_SLEEP_ENABLE  
_HAL_RCC_USART6_CLK_SLEEP_ENABLE  
_HAL_RCC_ADC1_CLK_SLEEP_ENABLE  
_HAL_RCC_SDIO_CLK_SLEEP_ENABLE  
_HAL_RCC_SPI1_CLK_SLEEP_ENABLE  
_HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM8_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM9_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM10_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM11_CLK_SLEEP_ENABLE  
_HAL_RCC_ADC2_CLK_SLEEP_ENABLE  
_HAL_RCC_ADC3_CLK_SLEEP_ENABLE  
_HAL_RCC_TIM1_CLK_SLEEP_DISABLE  
_HAL_RCC_USART1_CLK_SLEEP_DISABLE  
_HAL_RCC_USART6_CLK_SLEEP_DISABLE  
_HAL_RCC_ADC1_CLK_SLEEP_DISABLE  
_HAL_RCC_SDIO_CLK_SLEEP_DISABLE  
_HAL_RCC_SPI1_CLK_SLEEP_DISABLE  
_HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM8_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM9_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM10_CLK_SLEEP_DISABLE  
_HAL_RCC_TIM11_CLK_SLEEP_DISABLE  
_HAL_RCC_ADC2_CLK_SLEEP_DISABLE  
_HAL_RCC_ADC3_CLK_SLEEP_DISABLE  
RCC BitAddress AliasRegion  
RCC_OFFSET  
RCC_CR_OFFSET  
RCC_HSION_BIT_NUMBER  
RCC_CR_HSION_BB  
RCC_CSSON_BIT_NUMBER  
RCC_CR_CSSON_BB
```

RCC\_PLLON\_BIT\_NUMBER  
RCC\_CR\_PLLON\_BB  
RCC\_PLLI2SON\_BIT\_NUMBER  
RCC\_CR\_PLLI2SON\_BB  
RCC\_CFGR\_OFFSET  
RCC\_I2SSRC\_BIT\_NUMBER  
RCC\_CFGR\_I2SSRC\_BB  
RCC\_BDCR\_OFFSET  
RCC\_RTCEN\_BIT\_NUMBER  
RCC\_BDCR\_RTCEN\_BB  
RCC\_BDRST\_BIT\_NUMBER  
RCC\_BDCR\_BDRST\_BB  
RCC\_CSR\_OFFSET  
RCC\_LSION\_BIT\_NUMBER  
RCC\_CSR\_LSION\_BB  
RCC\_CR\_BYTE2\_ADDRESS  
RCC\_CIR\_BYTE1\_ADDRESS  
RCC\_CIR\_BYTE2\_ADDRESS  
RCC\_BDCR\_BYTE0\_ADDRESS  
RCC\_DBP\_TIMEOUT\_VALUE  
RCC\_LSE\_TIMEOUT\_VALUE  
HSE\_TIMEOUT\_VALUE  
HSI\_TIMEOUT\_VALUE  
LSI\_TIMEOUT\_VALUE  
PLL2S\_TIMEOUT\_VALUE

**RCC Exported Macros**

\_HAL\_RCC\_FSMC\_CLK\_SLEEP\_ENABLE  
\_HAL\_RCC\_FSMC\_CLK\_SLEEP\_DISABLE

**Flags**

RCC\_FLAG\_HSIRDY  
RCC\_FLAG\_HSERDY  
RCC\_FLAG\_PLLRDY  
RCC\_FLAG\_PLLI2SRDY  
RCC\_FLAG\_LSERDY  
RCC\_FLAG\_LSIRDY  
RCC\_FLAG\_BORRST

RCC\_FLAG\_PINRST  
 RCC\_FLAG\_PORRST  
 RCC\_FLAG\_SFTRST  
 RCC\_FLAG\_IWDGRST  
 RCC\_FLAG\_WWDGRST  
 RCC\_FLAG\_LPWRRST

**Flags Interrupts Management**

`__HAL_RCC_ENABLE_IT`

**Description:**

- Enable RCC interrupt (Perform Byte access to RCC\_CIR[14:8] bits to enable the selected interrupts).

**Parameters:**

- `__INTERRUPT__`: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt.
  - RCC\_IT\_LSERDY: LSE ready interrupt.
  - RCC\_IT\_HSIRDY: HSI ready interrupt.
  - RCC\_IT\_HSERDY: HSE ready interrupt.
  - RCC\_IT\_PLLRDY: Main PLL ready interrupt.
  - RCC\_IT\_PLLI2SRDY: PLLI2S ready interrupt.

`__HAL_RCC_DISABLE_IT`

**Description:**

- Disable RCC interrupt (Perform Byte access to RCC\_CIR[14:8] bits to disable the selected interrupts).

**Parameters:**

- `__INTERRUPT__`: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt.
  - RCC\_IT\_LSERDY: LSE ready interrupt.
  - RCC\_IT\_HSIRDY: HSI ready interrupt.
  - RCC\_IT\_HSERDY: HSE ready interrupt.
  - RCC\_IT\_PLLRDY: Main PLL ready interrupt.
  - RCC\_IT\_PLLI2SRDY: PLLI2S ready interrupt.

`__HAL_RCC_CLEAR_IT`

**Description:**

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC\_CIR[23:16]

bits to clear the selected interrupt pending bits.

#### Parameters:

- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt.
  - RCC\_IT\_LSERDY: LSE ready interrupt.
  - RCC\_IT\_HSIRDY: HSI ready interrupt.
  - RCC\_IT\_HSERDY: HSE ready interrupt.
  - RCC\_IT\_PLLRDY: Main PLL ready interrupt.
  - RCC\_IT\_PLLI2SRDY: PLLI2S ready interrupt.
  - RCC\_IT\_CSS: Clock Security System interrupt

### \_\_HAL\_RCC\_GET\_IT

#### Description:

- Check the RCC's interrupt has occurred or not.

#### Parameters:

- \_\_INTERRUPT\_\_: specifies the RCC interrupt source to check. This parameter can be one of the following values:
  - RCC\_IT\_LSIRDY: LSI ready interrupt.
  - RCC\_IT\_LSERDY: LSE ready interrupt.
  - RCC\_IT\_HSIRDY: HSI ready interrupt.
  - RCC\_IT\_HSERDY: HSE ready interrupt.
  - RCC\_IT\_PLLRDY: Main PLL ready interrupt.
  - RCC\_IT\_PLLI2SRDY: PLLI2S ready interrupt.
  - RCC\_IT\_CSS: Clock Security System interrupt

#### Return value:

- The: new state of \_\_INTERRUPT\_\_ (TRUE or FALSE).

### \_\_HAL\_RCC\_CLEAR\_RESET\_FLAGS

#### RCC\_FLAG\_MASK

#### Description:

- Check RCC flag is set or not.

#### Parameters:

- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - RCC\_FLAG\_HSIRDY: HSI oscillator clock ready.

- RCC\_FLAG\_HSERDY: HSE oscillator clock ready.
- RCC\_FLAG\_PLLRDY: Main PLL clock ready.
- RCC\_FLAG\_PLLI2SRDY: PLLI2S clock ready.
- RCC\_FLAG\_LSERDY: LSE oscillator clock ready.
- RCC\_FLAG\_LSIRDY: LSI oscillator clock ready.
- RCC\_FLAG\_BORRST: POR/PDR or BOR reset.
- RCC\_FLAG\_PINRST: Pin reset.
- RCC\_FLAG\_PORRST: POR/PDR reset.
- RCC\_FLAG\_SFTRST: Software reset.
- RCC\_FLAG\_IWDGRST: Independent Watchdog reset.
- RCC\_FLAG\_WWDGRST: Window Watchdog reset.
- RCC\_FLAG\_LPWRRST: Low Power reset.

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

`__HAL_RCC_GET_FLAG  
RCC_GET_PLL_OSCSOURCE`

**Get Clock source**

`__HAL_RCC_SYSCLK_CONFIG`

**Description:**

- Macro to configure the system clock source.

**Parameters:**

- `__RCC_SYSCLKSOURCE__`: specifies the system clock source. This parameter can be one of the following values:
  - `RCC_SYSCLKSOURCE_HSI`: HSI oscillator is used as system clock source.
  - `RCC_SYSCLKSOURCE_HSE`: HSE oscillator is used as system clock source.
  - `RCC_SYSCLKSOURCE_PLLCLK`: PLL output is used as system clock source.

`__HAL_RCC_GET_SYSCLK_SOURCE`

**Description:**

- Macro to get the clock source used as system clock.

**Return value:**

- The: clock source used as system clock. The returned value can be one of the following:
  - `RCC_SYSCLKSOURCE_STATUS_HSI`: HSI used as system clock.

- RCC\_SYSCLKSOURCE\_STATUS\_HSE: HSE used as system clock.
- RCC\_SYSCLKSOURCE\_STATUS\_PLLCLK: PLL used as system clock.

`__HAL_RCC_GET_PLL_OSCSOURCE`

**Description:**

- Macro to get the oscillator used as PLL clock source.

**Return value:**

- The oscillator used as PLL clock source. The returned value can be one of the following:
  - RCC\_PLLSOURCE\_HSI: HSI oscillator is used as PLL clock source.
  - RCC\_PLLSOURCE\_HSE: HSE oscillator is used as PLL clock source.

**HSE Config**

`RCC_HSE_OFF`

`RCC_HSE_ON`

`RCC_HSE_BYPASS`

**HSE Configuration**

`__HAL_RCC_HSE_CONFIG`

**Description:**

- Macro to configure the External High Speed oscillator (HSE).

**Parameters:**

- `__STATE__`: specifies the new state of the HSE. This parameter can be one of the following values:
  - `RCC_HSE_OFF`: turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  - `RCC_HSE_ON`: turn ON the HSE oscillator.
  - `RCC_HSE_BYPASS`: HSE oscillator bypassed with external clock.

**Notes:**

- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass. After enabling the HSE (`RCC_HSE_ON` or `RCC_HSE_Bypass`), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the clock security system(CSS) was previously enabled you

---

have to enable it again after calling this function.

***HSI Config***`RCC_HSI_OFF``RCC_HSI_ON`***HSI Configuration***`__HAL_RCC_HSI_ENABLE`**Notes:**

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after 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). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This parameter can be: ENABLE or DISABLE. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

`__HAL_RCC_HSI_DISABLE``__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST`**Description:**

- Macro to adjust the Internal High Speed oscillator (HSI) calibration value.

**Parameters:**

- `__HSICalibrationValue__`: specifies the calibration trimming value. This parameter must be a number between 0 and 0x1F.

**Notes:**

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the

internal HSI RC.

### **I2S Clock Source**

`RCC_I2SCLKSOURCE_PLLI2S`

`RCC_I2SCLKSOURCE_EXT`

### **RTC Clock Configuration**

`_HAL_RCC_RTC_ENABLE`

`_HAL_RCC_RTC_DISABLE`

`_HAL_RCC_RTC_CLKPRESCALER`

#### **Notes:**

- These macros must be used only after the RTC clock source was selected.

#### **Description:**

- Macros to configure the RTC clock (RTCCLK).

#### **Parameters:**

- `_RTCCLKSource_`: specifies the RTC clock source. This parameter can be one of the following values:
  - `RCC_RTCCLKSOURCE_LSE`: LSE selected as RTC clock.
  - `RCC_RTCCLKSOURCE_LSI`: LSI selected as RTC clock.
  - `RCC_RTCCLKSOURCE_HSE_DIVx`: HSE clock divided by x selected as RTC clock, where x:[2,31]

#### **Notes:**

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it can't be changed unless the Backup domain is reset using `_HAL_RCC_BackupReset_RELEASE()` macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wake-up source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

`_HAL_RCC_RTC_CONFIG`

`_HAL_RCC_BACKUPRESET_FORC  
E`

#### **Notes:**

- This function resets the RTC peripheral

(including the backup registers) and the RTC clock source selection in RCC\_CSR register. The BKPSRAM is not affected by this reset.

`_HAL_RCC_BACKUPRESET_RELEASE`

***Interrupts***

`RCC_IT_LSIRDY`

`RCC_IT_LSERDY`

`RCC_IT_HSIRDY`

`RCC_IT_HSERDY`

`RCC_IT_PLLRDY`

`RCC_IT_PLLI2SRDY`

`RCC_IT_CSS`

***RCC Private macros to check input parameters***

`IS_RCC_OSCILLATORTYPE`

`IS_RCC_HSE`

`IS_RCC_LSE`

`IS_RCC_HSI`

`IS_RCC_LSI`

`IS_RCC_PLL`

`IS_RCC_PLLSOURCE`

`IS_RCC_SYSCLKSOURCE`

`IS_RCC_PLLM_VALUE`

`IS_RCC_PLLN_VALUE`

`IS_RCC_PLLP_VALUE`

`IS_RCC_PLLQ_VALUE`

`IS_RCC_HCLK`

`IS_RCC_CLOCKTYPE`

`IS_RCC_PCLK`

`IS_RCC_MCO`

`IS_RCC_MCO1SOURCE`

`IS_RCC_MCO2SOURCE`

`IS_RCC_MCODIV`

`IS_RCC_CALIBRATION_VALUE`

***LSE Config***

`RCC_LSE_OFF`

`RCC_LSE_ON`

**RCC\_LSE\_BYPASS****LSE Configuration****\_\_HAL\_RCC\_LSE\_CONFIG Description:**

- Macro to configure the External Low Speed oscillator (LSE).

**Parameters:**

- STATE: specifies the new state of the LSE. This parameter can be one of the following values:
  - RCC\_LSE\_OFF: turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - RCC\_LSE\_ON: turn ON the LSE oscillator.
  - RCC\_LSE\_BYPASS: LSE oscillator bypassed with external clock.

**Notes:**

- Transition LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL\_PWR\_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC\_LSE\_ON or RCC\_LSE\_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

**LSI Config****RCC\_LSI\_OFF****RCC\_LSI\_ON****LSI Configuration****\_\_HAL\_RCC\_LSI\_ENABLE****Notes:**

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

**\_\_HAL\_RCC\_LSI\_DISABLE****MCO1 Clock Source****RCC\_MCO1SOURCE\_HSI****RCC\_MCO1SOURCE\_LSE****RCC\_MCO1SOURCE\_HSE****RCC\_MCO1SOURCE\_PLLCLK****MCO2 Clock Source**

RCC\_MCO2SOURCE\_SYSCLK  
RCC\_MCO2SOURCE\_PLLI2SCLK  
RCC\_MCO2SOURCE\_HSE  
RCC\_MCO2SOURCE\_PLLCLK

***MCOx Clock Prescaler***

RCC\_MCODIV\_1  
RCC\_MCODIV\_2  
RCC\_MCODIV\_3  
RCC\_MCODIV\_4  
RCC\_MCODIV\_5

***MCO Index***

RCC\_MCO1  
RCC\_MCO2

***Oscillator Type***

RCC\_OSCILLATORTYPE\_NONE  
RCC\_OSCILLATORTYPE\_HSE  
RCC\_OSCILLATORTYPE\_HSI  
RCC\_OSCILLATORTYPE\_LSE  
RCC\_OSCILLATORTYPE\_LSI

***PLLP Clock Divider***

RCC\_PLLP\_DIV2  
RCC\_PLLP\_DIV4  
RCC\_PLLP\_DIV6  
RCC\_PLLP\_DIV8

***PLL Clock Source***

RCC\_PLLSOURCE\_HSI  
RCC\_PLLSOURCE\_HSE

***PLL Config***

RCC\_PLL\_NONE  
RCC\_PLL\_OFF  
RCC\_PLL\_ON

***PLL Configuration***

`_HAL_RCC_PLL_ENABLE`    **Notes:**

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source. The main PLL is disabled by

hardware when entering STOP and STANDBY modes.

`_HAL_RCC_PLL_DISABLE`

`_HAL_RCC_PLL_CONFIG`

**Description:**

- Macro to configure the main PLL clock source, multiplication and division factors.

**Parameters:**

- `_RCC_PLLSource`: specifies the PLL entry clock source. This parameter can be one of the following values:
  - `RCC_PLLSOURCE_HSI`: HSI oscillator clock selected as PLL clock entry
  - `RCC_PLLSOURCE_HSE`: HSE oscillator clock selected as PLL clock entry
- `_PLLM`: specifies the division factor for PLL VCO input clock. This parameter must be a number between `Min_Data = 2` and `Max_Data = 63`.
- `_PLLN`: specifies the multiplication factor for PLL VCO output clock. This parameter must be a number between `Min_Data = 192` and `Max_Data = 432`.
- `_PLLP`: specifies the division factor for main system clock (SYSCLK). This parameter must be a number in the range {2, 4, 6, or 8}.
- `_PLLQ`: specifies the division factor for OTG FS, SDIO and RNG clocks. This parameter must be a number between `Min_Data = 2` and `Max_Data = 15`.

**Notes:**

- This function must be used only when the main PLL is disabled.
- This clock source (`RCC_PLLSource`) is common for the main PLL and PLLI2S.
- You have to set the `PLLM` parameter correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.
- You have to set the `PLLN` parameter correctly to ensure that the VCO output frequency is between 192 and 432 MHz.
- If the USB OTG FS is used in your application, you have to set the `PLLQ` parameter correctly to have 48 MHz clock for the USB. However, the SDIO and RNG need a frequency lower than or equal to 48 MHz to work correctly.

**PLL I2S Configuration**

`_HAL_RCC_PLLI2S_ENABLE`

**Notes:**

- The PLLI2S is disabled by hardware when entering STOP and STANDBY modes.

`_HAL_RCC_PLLI2S_DISABLE`

`_HAL_RCC_PLLI2S_CONFIG`

**Description:**

- Macro to configure the PLLI2S clock multiplication and division factors .

**Parameters:**

- `__PLLI2SN__`: specifies the multiplication factor for PLLI2S VCO output clock This parameter must be a number between Min\_Data = 192 and Max\_Data = 432.
- `__PLLI2SR__`: specifies the division factor for I2S clock This parameter must be a number between Min\_Data = 2 and Max\_Data = 7.

**Notes:**

- This macro must be used only when the PLLI2S is disabled. PLLI2S clock source is common with the main PLL (configured in `HAL_RCC_ClockConfig()` API).
- You have to set the PLLI2SN parameter correctly to ensure that the VCO output frequency is between Min\_Data = 192 and Max\_Data = 432 MHz.
- You have to set the PLLI2SR parameter correctly to not exceed 192 MHz on the I2S clock frequency.

`__HAL_RCC_I2S_CONFIG`

**Description:**

- Macro to configure the I2S clock source (I2SCLK).

**Parameters:**

- `__SOURCE__`: specifies the I2S clock source. This parameter can be one of the following values:
  - `RCC_I2SCLKSOURCE_PLLI2S`: PLLI2S clock used as I2S clock source.
  - `RCC_I2SCLKSOURCE_EXT`: External clock mapped on the I2S\_CKIN pin used as I2S clock source.

**Notes:**

- This function must be called before enabling the I2S APB clock.

**RCC Private Constants**

`CLOCKSWITCH_TIMEOUT_VALUE`

`_MCO1_CLK_ENABLE`

`MCO1_GPIO_PORT`

`MCO1_PIN`

`_MCO2_CLK_ENABLE`

`MCO2_GPIO_PORT`

`MCO2_PIN`

**RTC Clock Source**

`RCC_RTCCLKSOURCE_LSE`

RCC\_RTCCLKSOURCE\_LSI  
RCC\_RTCCLKSOURCE\_HSE\_DIV2  
RCC\_RTCCLKSOURCE\_HSE\_DIV3  
RCC\_RTCCLKSOURCE\_HSE\_DIV4  
RCC\_RTCCLKSOURCE\_HSE\_DIV5  
RCC\_RTCCLKSOURCE\_HSE\_DIV6  
RCC\_RTCCLKSOURCE\_HSE\_DIV7  
RCC\_RTCCLKSOURCE\_HSE\_DIV8  
RCC\_RTCCLKSOURCE\_HSE\_DIV9  
RCC\_RTCCLKSOURCE\_HSE\_DIV10  
RCC\_RTCCLKSOURCE\_HSE\_DIV11  
RCC\_RTCCLKSOURCE\_HSE\_DIV12  
RCC\_RTCCLKSOURCE\_HSE\_DIV13  
RCC\_RTCCLKSOURCE\_HSE\_DIV14  
RCC\_RTCCLKSOURCE\_HSE\_DIV15  
RCC\_RTCCLKSOURCE\_HSE\_DIV16  
RCC\_RTCCLKSOURCE\_HSE\_DIV17  
RCC\_RTCCLKSOURCE\_HSE\_DIV18  
RCC\_RTCCLKSOURCE\_HSE\_DIV19  
RCC\_RTCCLKSOURCE\_HSE\_DIV20  
RCC\_RTCCLKSOURCE\_HSE\_DIV21  
RCC\_RTCCLKSOURCE\_HSE\_DIV22  
RCC\_RTCCLKSOURCE\_HSE\_DIV23  
RCC\_RTCCLKSOURCE\_HSE\_DIV24  
RCC\_RTCCLKSOURCE\_HSE\_DIV25  
RCC\_RTCCLKSOURCE\_HSE\_DIV26  
RCC\_RTCCLKSOURCE\_HSE\_DIV27  
RCC\_RTCCLKSOURCE\_HSE\_DIV28  
RCC\_RTCCLKSOURCE\_HSE\_DIV29  
RCC\_RTCCLKSOURCE\_HSE\_DIV30  
RCC\_RTCCLKSOURCE\_HSE\_DIV31

***System Clock Source***

RCC\_SYSCLKSOURCE\_HSI  
RCC\_SYSCLKSOURCE\_HSE  
RCC\_SYSCLKSOURCE\_PLLCLK

***System Clock Source Status***

RCC_SYSCLKSOURCE_STATUS_HSI	HSI used as system clock
RCC_SYSCLKSOURCE_STATUS_HSE	HSE used as system clock
RCC_SYSCLKSOURCE_STATUS_PLLCLK	PLL used as system clock

***System Clock Type***

RCC_CLOCKTYPE_SYSCLK
RCC_CLOCKTYPE_HCLK
RCC_CLOCKTYPE_PCLK1
RCC_CLOCKTYPE_PCLK2

## 34 HAL RCC Extension Driver

### 34.1 RCCEEx Firmware driver registers structures

#### 34.1.1 RCC\_PLLI2SInitTypeDef

##### Data Fields

- *uint32\_t PLLI2SN*
- *uint32\_t PLLI2SR*

##### 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
- ***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

#### 34.1.2 RCC\_PeriphCLKInitTypeDef

##### Data Fields

- *uint32\_t PeriphClockSelection*
- *RCC\_PLLI2SInitTypeDef PLLI2S*
- *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
- ***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 [\*\*RCCEx\\_TIM\\_PRescaler\\_Selection\*\*](#)

## 34.2 RCCEEx Firmware driver API description

### 34.2.1 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.

This section contains the following APIs:

- [\*HAL\\_RCCEEx\\_PeriphCLKConfig\(\)\*](#)
- [\*HAL\\_RCCEEx\\_GetPeriphCLKConfig\(\)\*](#)

#### 34.2.2 HAL\_RCCEEx\_PeriphCLKConfig

Function Name	<code>HAL_StatusTypeDef HAL_RCCEEx_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)</code>
Function Description	Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit:</b> pointer to an RCC_PeriphCLKInitTypeDef 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>• HAL status</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> </ul>

#### 34.2.3 HAL\_RCCEEx\_GetPeriphCLKConfig

Function Name	<code>void HAL_RCCEEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)</code>
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>

## 34.3 RCCEEx Firmware driver defines

### 34.3.1 RCCEEx

*AHB2 Force Release Reset*

`_HAL_RCC_DCMI_FORCE_RESET`

```
_HAL_RCC_DCMI_RELEASE_RESET  
_HAL_RCC_CRYPT_FORCE_RESET  
_HAL_RCC_HASH_FORCE_RESET  
_HAL_RCC_CRYPT_RELEASE_RESET  
_HAL_RCC_HASH_RELEASE_RESET
```

**AHB2 Peripheral Low Power Enable Disable**

```
_HAL_RCC_DCMI_CLK_SLEEP_ENABLE  
_HAL_RCC_DCMI_CLK_SLEEP_DISABLE  
_HAL_RCC_CRYPT_CLK_SLEEP_ENABLE  
_HAL_RCC_HASH_CLK_SLEEP_ENABLE  
_HAL_RCC_CRYPT_CLK_SLEEP_DISABLE  
_HAL_RCC_HASH_CLK_SLEEP_DISABLE
```

**RCC BitAddress AliasRegion**

```
PLL_TIMEOUT_VALUE
```

**RCCEx Exported Macros**

```
_HAL_RCC_ETHMAC_CLK_ENABLE  
_HAL_RCC_ETHMACTX_CLK_ENABLE  
_HAL_RCC_ETHMACRX_CLK_ENABLE  
_HAL_RCC_ETHMACPTP_CLK_ENABLE  
_HAL_RCC_ETHMAC_CLK_DISABLE  
_HAL_RCC_ETHMACTX_CLK_DISABLE  
_HAL_RCC_ETHMACRX_CLK_DISABLE  
_HAL_RCC_ETHMACPTP_CLK_DISABLE  
_HAL_RCC_ETH_CLK_ENABLE  
_HAL_RCC_ETH_CLK_DISABLE
```

**RCC Private macros to check input parameters**

```
IS_RCC_PERIPH_CLOCK  
IS_RCC_PLLI2SN_VALUE  
IS_RCC_PLLI2SR_VALUE
```

**RCC Extended MCOx Clock Config**

```
_HAL_RCC_MCO1_CONFIG   Description:
```

- Macro to configure the MCO1 clock.

**Parameters:**

- \_MCOCLKSOURCE\_: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC\_MCO1SOURCE\_HSI: HSI clock selected as MCO1 source

- RCC\_MCO1SOURCE\_LSE: LSE clock selected as MCO1 source
- RCC\_MCO1SOURCE\_HSE: HSE clock selected as MCO1 source
- RCC\_MCO1SOURCE\_PLLCLK: main PLL clock selected as MCO1 source
- \_MCODIV\_: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC\_MCODIV\_1: no division applied to MCOx clock
  - RCC\_MCODIV\_2: division by 2 applied to MCOx clock
  - RCC\_MCODIV\_3: division by 3 applied to MCOx clock
  - RCC\_MCODIV\_4: division by 4 applied to MCOx clock
  - RCC\_MCODIV\_5: division by 5 applied to MCOx clock

[\\_\\_HAL\\_RCC\\_MCO2\\_CONFIG](#)**Description:**

- Macro to configure the MCO2 clock.

**Parameters:**

- \_MCOCLKSOURCE\_: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC\_MCO2SOURCE\_SYSCLK: System clock (SYSCLK) selected as MCO2 source
  - RCC\_MCO2SOURCE\_PLLI2SCLK: PLLI2S clock selected as MCO2 source
  - RCC\_MCO2SOURCE\_HSE: HSE clock selected as MCO2 source
  - RCC\_MCO2SOURCE\_PLLCLK: main PLL clock selected as MCO2 source
- \_MCODIV\_: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC\_MCODIV\_1: no division applied to MCOx clock
  - RCC\_MCODIV\_2: division by 2 applied to MCOx clock
  - RCC\_MCODIV\_3: division by 3 applied to MCOx clock
  - RCC\_MCODIV\_4: division by 4 applied to MCOx clock
  - RCC\_MCODIV\_5: division by 5 applied to MCOx clock

**RCC Periph Clock Selection**

[RCC\\_PERIPHCLK\\_I2S](#)  
[RCC\\_PERIPHCLK\\_TIM](#)  
[RCC\\_PERIPHCLK\\_RTC](#)  
[RCC\\_PERIPHCLK\\_PLLI2S](#)

***RCC TIM PRescaler Selection***

RCC\_TIMPRES\_DESACTIVATED

RCC\_TIMPRES\_ACTIVATED

## 35 HAL RNG Generic Driver

### 35.1 RNG Firmware driver registers structures

#### 35.1.1 RNG\_HandleTypeDef

##### Data Fields

- *RNG\_TypeDef \* Instance*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_RNG\_StateTypeDef State*
- *uint32\_t RandomNumber*

##### 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
- ***uint32\_t RNG\_HandleTypeDef::RandomNumber***  
Last Generated RNG Data

### 35.2 RNG Firmware driver API description

#### 35.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using `__HAL_RCC_RNG_CLK_ENABLE()` macro in `HAL_RNG_MspInit()`.
2. Activate the RNG peripheral using `HAL_RNG_Init()` function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using `HAL_RNG_GenerateRandomNumber()` function.

#### 35.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the `RNG_InitTypeDef` and create the associated handle
- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP

This section contains the following APIs:

- [`HAL\_RNG\_Init\(\)`](#)
- [`HAL\_RNG\_DeInit\(\)`](#)

- `HAL_RNG_MspInit()`
- `HAL_RNG_MspDeInit()`

### 35.2.3 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- `HAL_RNG_GenerateRandomNumber()`
- `HAL_RNG_GenerateRandomNumber_IT()`
- `HAL_RNG_IRQHandler()`
- `HAL_RNG_GetRandomNumber()`
- `HAL_RNG_GetRandomNumber_IT()`
- `HAL_RNG_ReadLastRandomNumber()`
- `HAL_RNG_ReadyDataCallback()`
- `HAL_RNG_ErrorCallback()`

### 35.2.4 Peripheral State functions

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

This section contains the following APIs:

- `HAL_RNG_GetState()`

### 35.2.5 HAL\_RNG\_Init

Function Name	<code>HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)</code>
Function Description	Initializes the RNG peripheral and creates the associated handle.
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>• HAL status</li> </ul>

### 35.2.6 HAL\_RNG\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)</code>
Function Description	Deinitializes the RNG peripheral.
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>• HAL status</li> </ul>

### 35.2.7 HAL\_RNG\_MspInit

Function Name	<code>void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)</code>
Function Description	Initializes the RNG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng:</b> pointer to a RNG_HandleTypeDef structure that</li> </ul>

contains the configuration information for RNG.

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

### 35.2.8 HAL\_RNG\_MspDelInit

Function Name	<b>void HAL_RNG_MspDelInit (RNG_HandleTypeDef * hrng)</b>
---------------	---

Function Description	DeInitializes the RNG MSP.
----------------------	----------------------------

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>
---------------	--

### 35.2.9 HAL\_RNG\_GenerateRandomNumber

Function Name	<b>HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)</b>
---------------	--

Function Description	Generates a 32-bit random number.
----------------------	-----------------------------------

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> <li><b>random32bit:</b> pointer to generated random number variable if successful.</li> </ul>
------------	--

Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
---------------	--

Notes	<ul style="list-style-type: none"> <li>Each time the random number data is read the RNG_FLAG_DRDY flag is automatically cleared.</li> </ul>
-------	---

### 35.2.10 HAL\_RNG\_GenerateRandomNumber\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)</b>
---------------	---

Function Description	Generates a 32-bit random number in interrupt 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>HAL status</li> </ul>
---------------	--

### 35.2.11 HAL\_RNG\_IRQHandler

Function Name	<b>void HAL_RNG_IRQHandler (RNG_HandleTypeDef * 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 __HAL_RNG_CLEAR_IT(). The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used.</li> </ul>
-------	---

- In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG\_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using `__HAL_RNG_CLEAR_IT()`, then disable and enable the RNG peripheral to reinitialize and restart the RNG.
- User-written HAL\_RNG\_ErrorCallback() API is called once whether SEIS or CEIS are set.

### 35.2.12 HAL\_RNG\_GetRandomNumber

Function Name	<code>uint32_t HAL_RNG_GetRandomNumber (RNG_HandleTypeDef * hrng)</code>
Function Description	Returns generated random number in polling mode (Obsolete) Use <code>HAL_RNG_GenerateRandomNumber()</code> API instead.
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>• Random value</li> </ul>

### 35.2.13 HAL\_RNG\_GetRandomNumber\_IT

Function Name	<code>uint32_t HAL_RNG_GetRandomNumber_IT (RNG_HandleTypeDef * hrng)</code>
Function Description	Returns a 32-bit random number with interrupt enabled (Obsolete), Use <code>HAL_RNG_GenerateRandomNumber_IT()</code> API instead.
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>• 32-bit random number</li> </ul>

### 35.2.14 HAL\_RNG\_ReadLastRandomNumber

Function Name	<code>uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)</code>
Function Description	Read latest generated random number.
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>• random value</li> </ul>

### 35.2.15 HAL\_RNG\_ReadyDataCallback

Function Name	<code>void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)</code>
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> <li>• <b>random32bit:</b> generated random number.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 35.2.16 HAL\_RNG\_ErrorCallback

Function Name	<b>void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * 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>

### 35.2.17 HAL\_RNG\_GetState

Function Name	<b>HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * 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>• HAL state</li> </ul>

## 35.3 RNG Firmware driver defines

### 35.3.1 RNG

#### *RNG Interrupt definition*

RNG_IT_DRDY	Data Ready interrupt
RNG_IT_CEI	Clock error interrupt
RNG_IT_SEI	Seed error interrupt

#### *RNG Flag definition*

RNG_FLAG_DRDY	Data ready
RNG_FLAG_CECS	Clock error current status
RNG_FLAG_SECS	Seed error current status

#### *RNG Exported Macros*

`_HAL_RNG_RESET_HANDLE_STATE` **Description:**

- Reset RNG handle state.

**Parameters:**

- `_HANDLE_`: RNG Handle

**Return value:**

- None

`_HAL_RNG_ENABLE`

**Description:**

- Enables the RNG peripheral.

**Parameters:**

- `_HANDLE_`: RNG Handle

**Return value:**

- None

**Description:**

- Disables the RNG peripheral.

**Parameters:**

- `__HANDLE__`: RNG Handle

**Return value:**

- None

**Description:**

- Check the selected RNG flag status.

**Parameters:**

- `__HANDLE__`: RNG Handle
- `__FLAG__`: RNG flag This parameter can be one of the following values:
  - `RNG_FLAG_DRDY`: Data ready current status
  - `RNG_FLAG_CECS`: Clock error current status
  - `RNG_FLAG_SECS`: Seed error current status

**Return value:**

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

**Description:**

- Clears the selected RNG flag status.

**Parameters:**

- `__HANDLE__`: RNG handle
- `__FLAG__`: RNG flag to clear

**Return value:**

- None

**Notes:**

- WARNING: This is a dummy macro for HAL code alignment, flags `RNG_FLAG_DRDY`, `RNG_FLAG_CECS` and `RNG_FLAG_SECS` are read-only.

**Description:**

- Enables the RNG interrupts.

**Parameters:**

- `__HANDLE__`: RNG Handle

**Return value:**

- None

**Description:**

- Disables the RNG interrupts.

**Parameters:**

- `__HANDLE__`: RNG Handle

**Return value:**

- None

**`__HAL_RNG_GET_IT`****Description:**

- Checks whether the specified RNG interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: RNG Handle
- `__INTERRUPT__`: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - `RNG_IT_DRDY`: Data ready interrupt
  - `RNG_IT_CEI`: Clock error interrupt
  - `RNG_IT_SEI`: Seed error interrupt

**Return value:**

- The new state of `__INTERRUPT__` (SET or RESET).

**`__HAL_RNG_CLEAR_IT`****Description:**

- Clear the RNG interrupt status flags.

**Parameters:**

- `__HANDLE__`: RNG Handle
- `__INTERRUPT__`: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - `RNG_IT_CEI`: Clock error interrupt
  - `RNG_IT_SEI`: Seed error interrupt

**Return value:**

- None

**Notes:**

- `RNG_IT_DRDY` flag is read-only, reading `RNG_DR` register automatically clears `RNG_IT_DRDY`.

***RNG Private Constants*****`RNG_TIMEOUT_VALUE`**

***RNG Private Macros***

IS\_RNG\_IT  
IS\_RNG\_FLAG

## 36 HAL RTC Generic Driver

### 36.1 RTC Firmware driver registers structures

#### 36.1.1 RTC\_InitTypeDef

##### 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 = 0xFFFF
- *uint32\_t RTC\_InitTypeDef::OutPut*  
Specifies which signal will be routed to the RTC output. This parameter can be a value of [\*RTC\\_Output\\_selection\\_Definitions\*](#)
- *uint32\_t RTC\_InitTypeDef::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\*](#)

#### 36.1.2 RTC\_TimeTypeDef

##### Data Fields

- *uint8\_t Hours*
- *uint8\_t Minutes*
- *uint8\_t Seconds*
- *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
- ***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\*\*](#)

**36.1.3 RTC\_DateTypeDef****Data Fields**

- ***uint8\_t WeekDay***
- ***uint8\_t Month***
- ***uint8\_t Date***
- ***uint8\_t Year***

**Field Documentation**

- ***uint8\_t RTC\_DateTypeDef::WeekDay***  
Specifies the RTC Date WeekDay. This parameter can be a value of [\*\*RTC\\_WeekDay\\_Definitions\*\*](#)
- ***uint8\_t RTC\_DateTypeDef::Month***  
Specifies the RTC Date Month (in BCD format). This parameter can be a value of [\*\*RTC\\_Month\\_Date\\_Definitions\*\*](#)
- ***uint8\_t RTC\_DateTypeDef::Date***  
Specifies the RTC Date. This parameter must be a number between Min\_Data = 1 and Max\_Data = 31
- ***uint8\_t RTC\_DateTypeDef::Year***  
Specifies the RTC Date Year. This parameter must be a number between Min\_Data = 0 and Max\_Data = 99

**36.1.4 RTC\_AlarmTypeDef**

**Data Fields**

- *RTC\_TimeTypeDef AlarmTime*
- *uint32\_t AlarmMask*
- *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::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\*](#)

**36.1.5 RTC\_HandleTypeDef****Data Fields**

- *RTC\_TypeDef \* Instance*
- *RTC\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_RTCStateTypeDef State*

**Field Documentation**

- *RTC\_TypeDef\* RTC\_HandleTypeDef::Instance*  
Register base address
- *RTC\_InitTypeDef RTC\_HandleTypeDef::Init*  
RTC required parameters
- *HAL\_LockTypeDef RTC\_HandleTypeDef::Lock*  
RTC locking object
- *\_\_IO HAL\_RTCStateTypeDef RTC\_HandleTypeDef::State*  
Time communication state

## 36.2 RTC Firmware driver API description

### 36.2.1 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

### 36.2.2 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.

### 36.2.3 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 \_\_HAL\_RCC\_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.

### 36.2.4 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.

## 36.2.5 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 wake-up, 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-wake-up mode), by using the RTC alarm or the RTC wake-up events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wake-up from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

## 36.2.6 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 wake-up from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the

RTC\_TR and RTC\_DR shadow registers. The HAL\_RTC\_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

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

### 36.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- [\*HAL\\_RTC\\_SetTime\(\)\*](#)
- [\*HAL\\_RTC\\_GetTime\(\)\*](#)
- [\*HAL\\_RTC\\_SetDate\(\)\*](#)
- [\*HAL\\_RTC\\_GetDate\(\)\*](#)

### 36.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- [\*HAL\\_RTC\\_SetAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_SetAlarm\\_IT\(\)\*](#)
- [\*HAL\\_RTC\\_DeactivateAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_GetAlarm\(\)\*](#)
- [\*HAL\\_RTC\\_AlarmIRQHandler\(\)\*](#)
- [\*HAL\\_RTC\\_AlarmAEventCallback\(\)\*](#)
- [\*HAL\\_RTC\\_PollForAlarmAEvent\(\)\*](#)

### 36.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- [\*HAL\\_RTC\\_WaitForSynchro\(\)\*](#)

### 36.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

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

### 36.2.11 HAL\_RTC\_Init

Function Name      **HAL\_StatusTypeDef HAL\_RTC\_Init (RTC\_HandleTypeDef \*  
                      hrtc)**

Function Description      Initializes the RTC peripheral.

Parameters      • **hrtc:** pointer to a RTC\_HandleTypeDef structure that

contains the configuration information for RTC.

Return values • HAL status

### 36.2.12 HAL\_RTC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)</b>
Function Description	DeInitializes the RTC peripheral.
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	• HAL status
Notes	• This function doesn't reset the RTC Backup Data registers.

### 36.2.13 HAL\_RTC\_MspInit

Function Name	<b>void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)</b>
Function Description	Initializes 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	• None

### 36.2.14 HAL\_RTC\_MspDeInit

Function Name	<b>void HAL_RTC_MspDeInit (RTC_HandleTypeDef * 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	• None

### 36.2.15 HAL\_RTC\_SetTime

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * 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: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format</li> </ul>
Return values	• HAL status

### 36.2.16 HAL\_RTC\_GetTime

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef</b>
---------------	---

**\* 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: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.</li> </ul>

### 36.2.17 HAL\_RTC\_SetDate

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef</b> <b>* hrtc, RTC_DateTypeDef * sDate, uint32_t Format)</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: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.18 HAL\_RTC\_GetDate

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef</b> <b>* hrtc, RTC_DateTypeDef * sDate, uint32_t Format)</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: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.</li> </ul>

### 36.2.19 HAL\_RTC\_SetAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)</b>
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: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.20 HAL\_RTC\_SetAlarm\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)</b>
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: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.21 HAL\_RTC\_DeactivateAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (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: RTC_ALARM_A: AlarmRTC_ALARM_B: AlarmB</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.22 HAL\_RTC\_GetAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, 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</li> </ul>

- contains the configuration information for RTC.
  - **sAlarm:** Pointer to Date structure
  - **Alarm:** Specifies the Alarm. This parameter can be one of the following values: RTC\_ALARM\_A:  
AlarmRTC\_ALARM\_B: AlarmB
  - **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:  
RTC\_FORMAT\_BIN: Binary data  
formatRTC\_FORMAT\_BCD: BCD data format
- Return values**
- HAL status

### 36.2.23 HAL\_RTC\_AlarmIRQHandler

Function Name	<b>void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)</b>
Function Description	This function handles Alarm 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>

### 36.2.24 HAL\_RTC\_AlarmAEventCallback

Function Name	<b>void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Alarm A 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>

### 36.2.25 HAL\_RTC\_PollForAlarmAEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
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>• HAL status</li> </ul>

### 36.2.26 HAL\_RTC\_WaitForSynchro

Function Name	<b>HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)</b>
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>• HAL status</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 wake-up 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>
-------	---

### 36.2.27 HAL\_RTC\_GetState

Function Name	<code>HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)</code>
Function Description	Returns the RTC state.
Parameters	<ul style="list-style-type: none"> <li><code>hrtc</code>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

## 36.3 RTC Firmware driver defines

### 36.3.1 RTC

***RTC Alarm Date WeekDay Definitions***

`RTC_ALARMDATEWEEKDAYSEL_DATE`

`RTC_ALARMDATEWEEKDAYSEL_WEEKDAY`

***RTC Alarm Mask Definitions***

`RTC_ALARMMASK_NONE`

`RTC_ALARMMASK_DATEWEEKDAY`

`RTC_ALARMMASK_HOURS`

`RTC_ALARMMASK_MINUTES`

`RTC_ALARMMASK_SECONDS`

`RTC_ALARMMASK_ALL`

***RTC Alarms Definitions***

`RTC_ALARM_A`

`RTC_ALARM_B`

***RTC AM PM Definitions***

`RTC_HOURFORMAT12_AM`

`RTC_HOURFORMAT12_PM`

***RTC DayLight Saving Definitions***

`RTC_DAYLIGHTSAVING_SUB1H`

`RTC_DAYLIGHTSAVING_ADD1H`

`RTC_DAYLIGHTSAVING_NONE`

***RTC Exported Macros***

`_HAL_RTC_RESET_HANDLE_STATE`

**Description:**

- Reset RTC handle state.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_WRITEPROTECTION_DISABLE`

**Description:**

- Disable the write protection for RTC registers.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_WRITEPROTECTION_ENABLE`

**Description:**

- Enable the write protection for RTC registers.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_ALARMA_ENABLE`

**Description:**

- Enable the RTC ALARMA peripheral.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_ALARMA_DISABLE`

**Description:**

- Disable the RTC ALARMA peripheral.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

`__HAL_RTC_ALARMB_ENABLE`

**Return value:**

- None

**Description:**

- Enable the RTC ALARMB peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_ALARMB_DISABLE`

**Description:**

- Disable the RTC ALARMB peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_ALARM_ENABLE_IT`

**Description:**

- Enable the RTC Alarm interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `RTC_IT_ALRA`: Alarm A interrupt
  - `RTC_IT_ALRB`: Alarm B interrupt

**Return value:**

- None

`__HAL_RTC_ALARM_DISABLE_IT`

**Description:**

- Disable the RTC Alarm interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt sources to be enabled or disabled. This

parameter can be any combination of the following values:

- RTC\_IT\_ALRA: Alarm A interrupt
- RTC\_IT\_ALRB: Alarm B interrupt

**Return value:**

- None

**\_HAL\_RTC\_ALARM\_GET\_IT**

- Check whether the specified RTC Alarm interrupt has occurred or not.

**Parameters:**

- HANDLE: specifies the RTC handle.
- INTERRUPT: specifies the RTC Alarm interrupt to check. This parameter can be:
  - RTC\_IT\_ALRA: Alarm A interrupt
  - RTC\_IT\_ALRB: Alarm B interrupt

**Return value:**

- None

**\_HAL\_RTC\_ALARM\_GET\_FLAG**

- Get the selected RTC Alarm's flag status.

**Parameters:**

- HANDLE: specifies the RTC handle.
- FLAG: specifies the RTC Alarm Flag to check. This parameter can be:
  - RTC\_FLAG\_ALRAF
  - RTC\_FLAG\_ALRBF
  - RTC\_FLAG\_ALRAWF
  - RTC\_FLAG\_ALRBWF

**Return value:**

- None

**\_HAL\_RTC\_ALARM\_CLEAR\_FLAG****Description:**

- Clear the RTC Alarm's pending flags.

**Parameters:**

- HANDLE: specifies the RTC

handle.

- \_\_FLAG\_\_: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - RTC\_FLAG\_ALRAF
  - RTC\_FLAG\_ALRBF

#### **Return value:**

- None

#### **Description:**

- Check whether the specified RTC Alarm interrupt has been enabled or not.

#### **Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC\_IT\_ALRA: Alarm A interrupt
  - RTC\_IT\_ALRB: Alarm B interrupt

#### **Return value:**

- None

#### **Description:**

- Enable interrupt on the RTC Alarm associated Exti line.

#### **Return value:**

- None

#### **Description:**

- Disable interrupt on the RTC Alarm associated Exti line.

#### **Return value:**

- None

#### **Description:**

- Enable event on the RTC Alarm associated Exti line.

#### **Return value:**

- None.

#### **Description:**

- Disable event on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Enable falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Disable falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Enable rising edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Disable rising edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

**Description:**

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

`_HAL_RTC_ALARM_EXTI_CLEAR_FLAG`

**Description:**

- Clear the RTC Alarm associated Exti line flag.

**Return value:**

- None.

`_HAL_RTC_ALARM_EXTI_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on RTC Alarm associated Exti line.

**Return value:**

- None.

***RTC Flags Definitions***

`RTC_FLAG_TAMP1F`

`RTC_FLAG_TSOVF`

`RTC_FLAG_TSF`

`RTC_FLAG_WUTF`

`RTC_FLAG_ALRBF`

`RTC_FLAG_ALRAF`

`RTC_FLAG_INITF`

`RTC_FLAG_RSF`

`RTC_FLAG_INITS`

`RTC_FLAG_WUTWF`

`RTC_FLAG_ALRBWF`

`RTC_FLAG_ALRAWF`

***RTC Hour Formats***

`RTC_HOURFORMAT_24`

`RTC_HOURFORMAT_12`

***RTC Input Parameter Format Definitions***

`RTC_FORMAT_BIN`

`RTC_FORMAT_BCD`

***RTC Interrupts Definitions***

`RTC_IT_TS`

`RTC_IT_WUT`

`RTC_IT_ALRB`

`RTC_IT_ALRA`

`RTC_IT_TAMP`

`RTC_IT_TAMP1`

***RTC Private macros to check input parameters***

IS\_RTC\_HOUR\_FORMAT  
IS\_RTC\_OUTPUT  
IS\_RTC\_OUTPUT\_POL  
IS\_RTC\_OUTPUT\_TYPE  
IS\_RTC\_HOUR12  
IS\_RTC\_HOUR24  
IS\_RTC\_ASYNCH\_PREDIV  
IS\_RTC\_SYNCH\_PREDIV  
IS\_RTC\_MINUTES  
IS\_RTC\_SECONDS  
IS\_RTC\_HOURFORMAT12  
IS\_RTC\_DAYLIGHT\_SAVING  
IS\_RTC\_STORE\_OPERATION  
IS\_RTC\_FORMAT  
IS\_RTC\_YEAR  
IS\_RTC\_MONTH  
IS\_RTC\_DATE  
IS\_RTC\_WEEKDAY  
IS\_RTC\_ALARM\_DATE\_WEEKDAY\_DATE  
IS\_RTC\_ALARM\_DATE\_WEEKDAY\_WEEKDAY  
IS\_RTC\_ALARM\_DATE\_WEEKDAY\_SEL  
IS\_RTC\_ALARM\_MASK  
IS\_RTC\_ALARM

***RTC Month Date Definitions***

RTC\_MONTH\_JANUARY  
RTC\_MONTH\_FEBRUARY  
RTC\_MONTH\_MARCH  
RTC\_MONTH\_APRIl  
RTC\_MONTH\_MAY  
RTC\_MONTH\_JUNE  
RTC\_MONTH\_JULY  
RTC\_MONTH\_AUGUST  
RTC\_MONTH\_SEPTEMBER  
RTC\_MONTH\_OCTOBER  
RTC\_MONTH\_NOVEMBER

RTC\_MONTH\_DECEMBER  
***RTC Output Polarity Definitions***  
RTC\_OUTPUT\_POLARITY\_HIGH  
RTC\_OUTPUT\_POLARITY\_LOW  
***RTC Output Selection Definitions***  
RTC\_OUTPUT\_DISABLE  
RTC\_OUTPUT\_ALARMA  
RTC\_OUTPUT\_ALARMMB  
RTC\_OUTPUT\_WAKEUP  
***RTC Output Type ALARM OUT***  
RTC\_OUTPUT\_TYPE\_OPENDRAIN  
RTC\_OUTPUT\_TYPE\_PUSH\_PULL  
***RTC Private Constants***  
RTC\_TR\_RESERVED\_MASK  
RTC\_DR\_RESERVED\_MASK  
RTC\_INIT\_MASK  
RTC\_RSF\_MASK  
RTC\_FLAGS\_MASK  
RTC\_TIMEOUT\_VALUE  
RTC\_EXTI\_LINE\_ALARM\_EVENT External interrupt line 17 Connected to the RTC Alarm event  
***RTC Store Operation Definitions***  
RTC\_STOREOPERATION\_RESET  
RTC\_STOREOPERATION\_SET  
***RTC WeekDay Definitions***  
RTC\_WEEKDAY\_MONDAY  
RTC\_WEEKDAY\_TUESDAY  
RTC\_WEEKDAY\_WEDNESDAY  
RTC\_WEEKDAY\_THURSDAY  
RTC\_WEEKDAY\_FRIDAY  
RTC\_WEEKDAY\_SATURDAY  
RTC\_WEEKDAY\_SUNDAY

## 37 HAL RTC Extension Driver

### 37.1 RTCEEx Firmware driver registers structures

#### 37.1.1 RTC\_TamperTypeDef

##### Data Fields

- *uint32\_t Tamper*
- *uint32\_t PinSelection*
- *uint32\_t Trigger*

##### Field Documentation

- *uint32\_t RTC\_TamperTypeDef::Tamper*  
Specifies the Tamper Pin. This parameter can be a value of [\*RTCEEx\\_Tamper\\_Pins\\_Definitions\*](#)
- *uint32\_t RTC\_TamperTypeDef::PinSelection*  
Specifies the Tamper Pin. This parameter can be a value of [\*RTCEEx\\_Tamper\\_Pins\\_Selection\*](#)
- *uint32\_t RTC\_TamperTypeDef::Trigger*  
Specifies the Tamper Trigger. This parameter can be a value of [\*RTCEEx\\_Tamper\\_Trigger\\_Definitions\*](#)

### 37.2 RTCEEx Firmware driver API description

#### 37.2.1 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 Wake-up configuration

- To configure the RTC Wake-up Clock source and Counter use the HAL\_RTCEx\_SetWakeUpTimer() function. You can also configure the RTC Wake-up timer in interrupt mode using the HAL\_RTCEx\_SetWakeUpTimer\_IT() function.
- To read the RTC Wake-up Counter register, use the HAL\_RTCEx\_GetWakeUpTimer() function.

##### TimeStamp configuration

- Configure the RTC\_AFx trigger and enable the RTC TimeStamp using the HAL\_RTCEx\_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL\_RTCEx\_SetTimeStamp\_IT() function.

- To read the RTC TimeStamp Time and Date register, use the `HAL_RTCEx_GetTimeStamp()` function.
- 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_RTCEx_SetTimeStamp()` or `HAL_RTCEx_SetTimeStamp_IT()` function.

### Tamper configuration

- Enable the RTC Tamper and configure the trigger using the `HAL_RTCEx_SetTamper()` function. You can configure RTC Tamper in interrupt mode using `HAL_RTCEx_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_RTCEx_SetTamper()` or `HAL_RTCEx_SetTamper_IT()` function.

### Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the `HAL_RTCEx_BKUPWrite()` function.
- To read the RTC Backup Data registers, use the `HAL_RTCEx_BKUPRead()` function.

## 37.2.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- `HAL_RTCEx_SetTimeStamp()`
- `HAL_RTCEx_SetTimeStamp_IT()`
- `HAL_RTCEx_DeactivateTimeStamp()`
- `HAL_RTCEx_GetTimeStamp()`
- `HAL_RTCEx_SetTamper()`
- `HAL_RTCEx_SetTamper_IT()`
- `HAL_RTCEx_DeactivateTamper()`
- `HAL_RTCEx_TamperTimeStampIRQHandler()`
- `HAL_RTCEx_TimeStampEventCallback()`
- `HAL_RTCEx_Tamper1EventCallback()`
- `HAL_RTCEx_PollForTimeStampEvent()`
- `HAL_RTCEx_PollForTamper1Event()`

## 37.2.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- `HAL_RTCEx_SetWakeUpTimer()`
- `HAL_RTCEx_SetWakeUpTimer_IT()`
- `HAL_RTCEx_DeactivateWakeUpTimer()`
- `HAL_RTCEx_GetWakeUpTimer()`
- `HAL_RTCEx_WakeUpTimerIRQHandler()`
- `HAL_RTCEx_WakeUpTimerEventCallback()`

- [\*HAL\\_RTCEEx\\_PollForWakeUpTimerEvent\(\)\*](#)

### 37.2.4 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
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC\_CALIB).
- Deactivate the Calibration Pinout (RTC\_CALIB).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.

This section contains the following APIs:

- [\*HAL\\_RTCEEx\\_BKUPWrite\(\)\*](#)
- [\*HAL\\_RTCEEx\\_BKUPRead\(\)\*](#)
- [\*HAL\\_RTCEEx\\_SetCoarseCalib\(\)\*](#)
- [\*HAL\\_RTCEEx\\_DeactivateCoarseCalib\(\)\*](#)
- [\*HAL\\_RTCEEx\\_SetCalibrationOutPut\(\)\*](#)
- [\*HAL\\_RTCEEx\\_DeactivateCalibrationOutPut\(\)\*](#)
- [\*HAL\\_RTCEEx\\_SetRefClock\(\)\*](#)
- [\*HAL\\_RTCEEx\\_DeactivateRefClock\(\)\*](#)

### 37.2.5 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- [\*HAL\\_RTCEEx\\_AlarmBEventCallback\(\)\*](#)
- [\*HAL\\_RTCEEx\\_PollForAlarmBEvent\(\)\*](#)

### 37.2.6 HAL\_RTCEEx\_SetTimeStamp

**Function Name** [\*HAL\\_StatusTypeDef HAL\\_RTCEEx\\_SetTimeStamp  
\(RTC\\_HandleTypeDef \\* hrtc, uint32\\_t TimeStampEdge,  
uint32\\_t RTC\\_TimeStampPin\)\*](#)

**Function Description** Sets TimeStamp.

**Parameters**

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: RTC\_TIMESTAMPEDGE\_RISING: the Time stamp event occurs on the rising edge of the related pin.RTC\_TIMESTAMPEDGE\_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC\_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values: RTC\_TIMESTAMPPIN\_DEFAULT: PC13 is selected as RTC TimeStamp Pin.RTC\_TIMESTAMPPIN\_POS1: PI8 is

selected as RTC TimeStamp Pin.

Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the TimeStamp feature.</li> </ul>

### 37.2.7 HAL\_RTCEx\_SetTimeStamp\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_tTimeStampEdge, uint32_t RTC_TimeStampPin)</b>
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> <li>• <b>TimeStampEdge:</b> Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.</li> <li>• <b>RTC_TimeStampPin:</b> Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin.RTC_TIMESTAMPPIN_POS1: PI8 is selected as RTC TimeStamp Pin.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the TimeStamp feature.</li> </ul>

### 37.2.8 HAL\_RTCEx\_DeactivateTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)</b>
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>• HAL status</li> </ul>

### 37.2.9 HAL\_RTCEx\_GetTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)</b>
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</li> </ul>

parameter can be one of the following values:  
**RTC\_FORMAT\_BIN**: Binary data format  
**RTC\_FORMAT\_BCD**: BCD data format

- Return values
- HAL status

### 37.2.10 HAL\_RTCEx\_SetTamper

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</b>
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>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we disable the tamper interrupt for all tampers.</li> </ul>

### 37.2.11 HAL\_RTCEx\_SetTamper\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)</b>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• By calling this API we force the tamper interrupt for all tampers.</li> </ul>

### 37.2.12 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.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.13 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 <b>37.2.14 HAL_RTCEx_TimeStampEventCallback</b> Function Name Function Description Parameters Return values	<ul style="list-style-type: none"> <li>• None</li> </ul> <p><b>void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)</b></p> <p>TimeStamp callback.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>37.2.15 HAL_RTCEx_Tamper1EventCallback</b> Function Name Function Description Parameters Return values	<p><b>void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)</b></p> <p>Tamper 1 callback.</p> <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> <ul style="list-style-type: none"> <li>• None</li> </ul>
<b>37.2.16 HAL_RTCEx_PollForTimeStampEvent</b> Function Name Function Description Parameters Return values	<p><b>HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b></p> <p>This function handles TimeStamp polling request.</p> <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> <ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>37.2.17 HAL_RTCEx_PollForTamper1Event</b> Function Name Function Description Parameters Return values	<p><b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b></p> <p>This function handles Tamper1 Polling.</p> <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> <ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>37.2.18 HAL_RTCEx_SetWakeUpTimer</b> Function Name Function Description Parameters	<p><b>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)</b></p> <p>Sets wake up timer.</p> <ul style="list-style-type: none"> <li>• <b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that</li> </ul>

	contains the configuration information for RTC.
•	<b>WakeUpCounter:</b> Wake up counter
•	<b>WakeUpClock:</b> Wake up clock

Return values      • HAL status

### 37.2.19 HAL\_RTCEx\_SetWakeUpTimer\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, 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	• HAL status

### 37.2.20 HAL\_RTCEx\_DeactivateWakeUpTimer

Function Name	<b>uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)</b>
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	• HAL status

### 37.2.21 HAL\_RTCEx\_GetWakeUpTimer

Function Name	<b>uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)</b>
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	• Counter value

### 37.2.22 HAL\_RTCEx\_WakeUpTimerIRQHandler

Function Name	<b>void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)</b>
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	• None

### 37.2.23 HAL\_RTCEx\_WakeUpTimerEventCallback

Function Name	<b>void HAL_RTCEx_WakeUpTimerEventCallback</b>
---------------	--

**(RTC\_HandleTypeDef \* hrtc)**

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>

**37.2.24 HAL\_RTCEx\_PollForWakeUpTimerEvent**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
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>• HAL status</li> </ul>

**37.2.25 HAL\_RTCEx\_BKUPWrite**

Function Name	<b>void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)</b>
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 parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.</li> <li>• <b>Data:</b> Data to be written in the specified RTC Backup data register.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**37.2.26 HAL\_RTCEx\_BKUPRead**

Function Name	<b>uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)</b>
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 RTC_HandleTypeDef 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	<ul style="list-style-type: none"> <li>• Read value</li> </ul>

**37.2.27 HAL\_RTCEx\_SetCoarseCalib**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetCoarseCalib (RTC_HandleTypeDef * hrtc, uint32_t CalibSign, uint32_t Value)</b>
---------------	--

Function Description	Sets 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> <li><b>CalibSign:</b> Specifies the sign of the coarse calibration value. This parameter can be one of the following values : RTC_CALIBSIGN_POSITIVE: The value sign is positiveRTC_CALIBSIGN_NEGATIVE: The value sign is negative</li> <li><b>Value:</b> value of coarse calibration expressed in ppm (coded on 5 bits).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
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>

### 37.2.28 HAL\_RTCEx\_DeactivateCoarseCalib

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateCoarseCalib (RTC_HandleTypeDef * hrtc)</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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 37.2.29 HAL\_RTCEx\_SetCalibrationOutPut

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc)</b>
Function Description	Configure the Calibration Pinout (RTC_CALIB).
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> : RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 37.2.30 HAL\_RTCEx\_DeactivateCalibrationOutPut

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)</b>
Function Description	Deactivates the Calibration Pinout (RTC_CALIB).
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>HAL status</li> </ul>

### 37.2.31 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.32 HAL\_RTCEx\_DeactivateRefClock

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)</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>• HAL status</li> </ul>

### 37.2.33 HAL\_RTCEx\_AlarmBEventCallback

Function Name	<b>void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)</b>
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>

### 37.2.34 HAL\_RTCEx\_PollForAlarmBEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
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>• HAL status</li> </ul>

## 37.3 RTCEx Firmware driver defines

### 37.3.1 RTCEx

#### *RTC Backup Registers Definitions*

RTC\_BKP\_DR0  
RTC\_BKP\_DR1  
RTC\_BKP\_DR2  
RTC\_BKP\_DR3  
RTC\_BKP\_DR4  
RTC\_BKP\_DR5

RTC\_BKP\_DR6  
RTC\_BKP\_DR7  
RTC\_BKP\_DR8  
RTC\_BKP\_DR9  
RTC\_BKP\_DR10  
RTC\_BKP\_DR11  
RTC\_BKP\_DR12  
RTC\_BKP\_DR13  
RTC\_BKP\_DR14  
RTC\_BKP\_DR15  
RTC\_BKP\_DR16  
RTC\_BKP\_DR17  
RTC\_BKP\_DR18  
RTC\_BKP\_DR19

***RTC Calibration***

`__HAL_RTC_COARSE_CALIB_ENABLE`

**Description:**

- Enable the Coarse calibration process.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_COARSE_CALIB_DISABLE`

**Description:**

- Disable the Coarse calibration process.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_CALIBRATION_OUTPUT_ENABLE`

**Description:**

- Enable the RTC calibration output.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_CALIBRATION_OUTPUT_DISABLE`

- Disable the calibration output.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_CLOCKREF_DETECTION_ENABLE`

- Enable the clock reference detection.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_CLOCKREF_DETECTION_DISABLE`

- Disable the clock reference detection.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

***RTC Digital Calib Definitions***

`RTC_CALIBSIGN_POSITIVE`  
`RTC_CALIBSIGN_NEGATIVE`

***Private macros to check input parameters***

`IS_RTC_BKP`  
`IS_TIMESTAMP_EDGE`  
`IS_RTC_TAMPER_PIN`  
`IS_RTC_TIMESTAMP_PIN`  
`IS_RTC_TAMPER_TRIGGER`  
`IS_RTC_WAKEUP_CLOCK`  
`IS_RTC_WAKEUP_COUNTER`  
`IS_RTC_CALIB_SIGN`  
`IS_RTC_CALIB_VALUE`  
`IS_RTC_TAMPER`

**RTCEx Private Constants**

RTC EXTI LINE TAMPER\_TIMESTAMP\_EVENT External interrupt line 21 Connected to the RTC Tamper and Time Stamp events

RTC EXTI LINE WAKEUPTIMER\_EVENT External interrupt line 22 Connected to the RTC Wake-up event

**RTC Tamper**

`_HAL_RTC_TAMPER1_ENABLE`

**Description:**

- Enable the RTC Tamper1 input detection.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_TAMPER1_DISABLE`

**Description:**

- Disable the RTC Tamper1 input detection.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

**Return value:**

- None

`_HAL_RTC_TAMPER_GET_IT`

**Description:**

- Check whether the specified RTC Tamper interrupt has occurred or not.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.
- `_INTERRUPT_`: specifies the RTC Tamper interrupt to check. This parameter can be:
  - `RTC_IT_TAMP1`

**Return value:**

- None

`_HAL_RTC_TAMPER_GET_IT_SOURCE`

**Description:**

- Check whether the specified RTC Tamper interrupt has been enabled or not.

**Parameters:**

- `_HANDLE_`: specifies the RTC handle.

- \_\_INTERRUPT\_\_: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC\_IT\_TAMP: Tamper interrupt

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_TAMPER\\_GET\\_FLAG](#)**Description:**

- Get the selected RTC Tamper's flag status.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_FLAG\_\_: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - RTC\_FLAG\_TAMP1F

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_TAMPER\\_CLEAR\\_FLAG](#)**Description:**

- Clear the RTC Tamper's pending flags.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_FLAG\_\_: specifies the RTC Tamper Flag to clear. This parameter can be:
  - RTC\_FLAG\_TAMP1F

**Return value:**

- None

***RTC Tamper Pins Definitions***[RTC\\_TAMPER\\_1](#)***RTC tamper Pins Selection***[RTC\\_TAMPERPIN\\_DEFAULT](#)[RTC\\_TAMPERPIN\\_POS1](#)***EXTI RTC Tamper Timestamp EXTI***[\\_\\_HAL\\_RTC\\_TAMPER\\_TIMESTAMP\\_EXTI\\_ENABLE\\_IT](#)**Description:**

- Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_TAMPER\\_TIMESTAMP\\_EXTI\\_DISABLE\\_IT](#)**Description:**

- Disable interrupt on the RTC Tamper

and Timestamp associated Exti line.

**Return value:**

- None

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_ENABLE_EVENT`

**Description:**

- Enable event on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_DISABLE_EVENT`

**Description:**

- Disable event on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_ENABLE_FALLING_EDGE`

**Description:**

- Enable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_DISABLE_FALLING_EDGE`

**Description:**

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_ENABLE_RISING_EDGE`

**Description:**

- Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_DISABLE_RISING_EDGE`

**Description:**

- Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_ENABLE_RISING_FALLING_EDGE`

**Description:**

- Enable rising & falling edge trigger on the RTC Tamper and Timestamp

associated Exti line.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_DISABLE_RISING_FALLING_EDGE`

**Description:**

- Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_GET_FLAG`

**Description:**

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_CLEAR_FLAG`

**Description:**

- Clear the RTC Tamper and Timestamp associated Exti line flag.

**Return value:**

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI  
_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

### *RTC Tamper Triggers Definitions*

`RTC_TAMPERTRIGGER_RISINGEDGE`

`RTC_TAMPERTRIGGER_FALLINGEDGE`

### *RTC Timestamp*

`__HAL_RTC_TIMESTAMP_ENABLE`

**Description:**

- Enable the RTC TimeStamp peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

`__HAL_RTC_TIMESTAMP_DISABLE`

**Description:**

- Disable the RTC TimeStamp peripheral.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.

**Return value:**

- None

\_\_HAL\_RTC\_TIMESTAMP\_ENABLE\_IT

**Description:**

- Enable the RTC TimeStamp interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
  - RTC\_IT\_TS: TimeStamp interrupt

**Return value:**

- None

\_\_HAL\_RTC\_TIMESTAMP\_DISABLE\_IT

**Description:**

- Disable the RTC TimeStamp interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
  - RTC\_IT\_TS: TimeStamp interrupt

**Return value:**

- None

\_\_HAL\_RTC\_TIMESTAMP\_GET\_IT

**Description:**

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC TimeStamp interrupt to check.

This parameter can be:

- RTC\_IT\_TS: TimeStamp interrupt

**Return value:**

- None

`__HAL_RTC_TIMESTAMP_GET_IT_SOURCE`

- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  - RTC\_IT\_TS: TimeStamp interrupt

**Return value:**

- None

`__HAL_RTC_TIMESTAMP_GET_FLAG`

- Get the selected RTC TimeStamp's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC TimeStamp flag to check. This parameter can be:
  - RTC\_FLAG\_TSF
  - RTC\_FLAG\_TSOVF

**Return value:**

- None

`__HAL_RTC_TIMESTAMP_CLEAR_FLAG`

- Clear the RTC Time Stamp's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
  - RTC\_FLAG\_TSF

**Return value:**

- None

***RTC TimeStamp Pins Selection***

RTC\_TIMESTAMPIN\_DEFAULT

RTC\_TIMESTAMPIN\_POS1

***RTC TimeStamp Edges Definitions***

RTC\_TIMESTAMPEDGE\_RISING

RTC\_TIMESTAMPEDGE\_FALLING

***RTC WakeUp Timer***`__HAL_RTC_WAKEUPTIMER_ENABLE`**Description:**

- Enable the RTC WakeUp Timer peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Disable the RTC Wake-up Timer peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Enable the RTC WakeUpTimer interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
  - `RTC_IT_WUT`: WakeUpTimer A interrupt

**Return value:**

- None

**Description:**

- Disable the RTC WakeUpTimer interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
  - RTC\_IT\_WUT: WakeUpTimer A interrupt

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_WAKEUPTIMER\\_GET\\_IT](#)**Description:**

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC WakeUpTimer interrupt to check. This parameter can be:
  - RTC\_IT\_WUT: WakeUpTimer A interrupt

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_WAKEUPTIMER\\_GET\\_IT\\_SOURCE](#)**Description:**

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_INTERRUPT\_\_: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
  - RTC\_IT\_WUT: WakeUpTimer interrupt

**Return value:**

- None

[\\_\\_HAL\\_RTC\\_WAKEUPTIMER\\_GET\\_FLAG](#)**Description:**

- Get the selected RTC WakeUpTimer's flag status.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC

handle.

- FLAG: specifies the RTC WakeUpTimer Flag to check. This parameter can be:
  - RTC\_FLAG\_WUTF
  - RTC\_FLAG\_WUTWF

**Return value:**

- None

\_HAL\_RTC\_WAKEUPTIMER\_CLEAR\_FLAG

**Description:**

- Clear the RTC Wake Up timer's pending flags.

**Parameters:**

- HANDLE: specifies the RTC handle.
- FLAG: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
  - RTC\_FLAG\_WUTF

**Return value:**

- None

\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_IT

**Description:**

- Enable interrupt on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None

\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_IT

**Description:**

- Disable interrupt on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None

\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_EVENT

**Description:**

- Enable event on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_EVENT

**Description:**

- Disable event on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE  
_FALLING_EDGE`

**Description:**

- Enable falling edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE  
_FALLING_EDGE`

**Description:**

- Disable falling edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE  
_RISING_EDGE`

**Description:**

- Enable rising edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE  
_RISING_EDGE`

**Description:**

- Disable rising edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE  
_RISING_FALLING_EDGE`

**Description:**

- Enable rising & falling edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE  
_RISING_FALLING_EDGE`

**Description:**

- Disable rising & falling edge trigger on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_GET_FL  
AG`

**Description:**

- Check whether the RTC Wake-up Timer associated Exti line interrupt flag is set or not.

**Return value:**

`__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG`

- Line: Status.

**Description:**

- Clear the RTC Wake-up Timer associated Exti line flag.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_GENERATE_SWIT`

**Description:**

- Generate a Software interrupt on the RTC Wake-up Timer associated Exti line.

**Return value:**

- None.

***RTC Wake-up Timer Definitions***

`RTC_WAKEUPCLOCK_RTCCLK_DIV16`  
`RTC_WAKEUPCLOCK_RTCCLK_DIV8`  
`RTC_WAKEUPCLOCK_RTCCLK_DIV4`  
`RTC_WAKEUPCLOCK_RTCCLK_DIV2`  
`RTC_WAKEUPCLOCK_CK_SPRE_16BITS`  
`RTC_WAKEUPCLOCK_CK_SPRE_17BITS`

## 38 HAL SD Generic Driver

### 38.1 SD Firmware driver registers structures

#### 38.1.1 SD\_HandleTypeDef

##### Data Fields

- *SD\_TypeDef \* Instance*
- *SD\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *uint32\_t CardType*
- *uint32\_t RCA*
- *uint32\_t CSD*
- *uint32\_t CID*
- *\_IO uint32\_t SdTransferCplt*
- *\_IO uint32\_t SdTransferErr*
- *\_IO uint32\_t DmaTransferCplt*
- *\_IO uint32\_t SdOperation*
- *DMA\_HandleTypeDef \* hdmarx*
- *DMA\_HandleTypeDef \* hdmatx*

##### Field Documentation

- ***SD\_TypeDef\* SD\_HandleTypeDef::Instance***  
SDIO register base address
- ***SD\_InitTypeDef SD\_HandleTypeDef::Init***  
SD required parameters
- ***HAL\_LockTypeDef SD\_HandleTypeDef::Lock***  
SD locking object
- ***uint32\_t SD\_HandleTypeDef::CardType***  
SD card type
- ***uint32\_t SD\_HandleTypeDef::RCA***  
SD relative card address
- ***uint32\_t SD\_HandleTypeDef::CSD[4]***  
SD card specific data table
- ***uint32\_t SD\_HandleTypeDef::CID[4]***  
SD card identification number table
- ***\_IO uint32\_t SD\_HandleTypeDef::SdTransferCplt***  
SD transfer complete flag in non blocking mode
- ***\_IO uint32\_t SD\_HandleTypeDef::SdTransferErr***  
SD transfer error flag in non blocking mode
- ***\_IO uint32\_t SD\_HandleTypeDef::DmaTransferCplt***  
SD DMA transfer complete flag
- ***\_IO uint32\_t SD\_HandleTypeDef::SdOperation***  
SD transfer operation (read/write)
- ***DMA\_HandleTypeDef\* SD\_HandleTypeDef::hdmarx***  
SD Rx DMA handle parameters
- ***DMA\_HandleTypeDef\* SD\_HandleTypeDef::hdmatx***  
SD Tx DMA handle parameters

### 38.1.2 HAL\_SD\_CSDTypedef

#### Data Fields

- `__IO uint8_t CSDStruct`
- `__IO uint8_t SysSpecVersion`
- `__IO uint8_t Reserved1`
- `__IO uint8_t TAAC`
- `__IO uint8_t NSAC`
- `__IO uint8_t MaxBusClkFrec`
- `__IO uint16_t CardComdClasses`
- `__IO uint8_t RdBlockLen`
- `__IO uint8_t PartBlockRead`
- `__IO uint8_t WrBlockMisalign`
- `__IO uint8_t RdBlockMisalign`
- `__IO uint8_t DSRImpl`
- `__IO uint8_t Reserved2`
- `__IO uint32_t DeviceSize`
- `__IO uint8_t MaxRdCurrentVDDMin`
- `__IO uint8_t MaxRdCurrentVDDMax`
- `__IO uint8_t MaxWrCurrentVDDMin`
- `__IO uint8_t MaxWrCurrentVDDMax`
- `__IO uint8_t DeviceSizeMul`
- `__IO uint8_t EraseGrSize`
- `__IO uint8_t EraseGrMul`
- `__IO uint8_t WrProtectGrSize`
- `__IO uint8_t WrProtectGrEnable`
- `__IO uint8_t ManDefIECC`
- `__IO uint8_t WrSpeedFact`
- `__IO uint8_t MaxWrBlockLen`
- `__IO uint8_t WriteBlockPaPartial`
- `__IO uint8_t Reserved3`
- `__IO uint8_t ContentProtectAppli`
- `__IO uint8_t FileFormatGrouop`
- `__IO uint8_t CopyFlag`
- `__IO uint8_t PermWrProtect`
- `__IO uint8_t TempWrProtect`
- `__IO uint8_t FileFormat`
- `__IO uint8_t ECC`
- `__IO uint8_t CSD_CRC`
- `__IO uint8_t Reserved4`

#### Field Documentation

- `__IO uint8_t HAL_SD_CSDTypedef::CSDStruct`  
CSD structure
- `__IO uint8_t HAL_SD_CSDTypedef::SysSpecVersion`  
System specification version
- `__IO uint8_t HAL_SD_CSDTypedef::Reserved1`  
Reserved

- **`_IO uint8_t HAL_SD_CSDTypeDef::TAAC`**  
Data read access time 1
- **`_IO uint8_t HAL_SD_CSDTypeDef::NSAC`**  
Data read access time 2 in CLK cycles
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxBusClkFrec`**  
Max. bus clock frequency
- **`_IO uint16_t HAL_SD_CSDTypeDef::CardComdClasses`**  
Card command classes
- **`_IO uint8_t HAL_SD_CSDTypeDef::RdBlockLen`**  
Max. read data block length
- **`_IO uint8_t HAL_SD_CSDTypeDef::PartBlockRead`**  
Partial blocks for read allowed
- **`_IO uint8_t HAL_SD_CSDTypeDef::WrBlockMisalign`**  
Write block misalignment
- **`_IO uint8_t HAL_SD_CSDTypeDef::RdBlockMisalign`**  
Read block misalignment
- **`_IO uint8_t HAL_SD_CSDTypeDef::DSRImpl`**  
DSR implemented
- **`_IO uint8_t HAL_SD_CSDTypeDef::Reserved2`**  
Reserved
- **`_IO uint32_t HAL_SD_CSDTypeDef::DeviceSize`**  
Device Size
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxRdCurrentVDDMin`**  
Max. read current @ VDD min
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxRdCurrentVDDMax`**  
Max. read current @ VDD max
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxWrCurrentVDDMin`**  
Max. write current @ VDD min
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxWrCurrentVDDMax`**  
Max. write current @ VDD max
- **`_IO uint8_t HAL_SD_CSDTypeDef::DeviceSizeMul`**  
Device size multiplier
- **`_IO uint8_t HAL_SD_CSDTypeDef::EraseGrSize`**  
Erase group size
- **`_IO uint8_t HAL_SD_CSDTypeDef::EraseGrMul`**  
Erase group size multiplier
- **`_IO uint8_t HAL_SD_CSDTypeDef::WrProtectGrSize`**  
Write protect group size
- **`_IO uint8_t HAL_SD_CSDTypeDef::WrProtectGrEnable`**  
Write protect group enable
- **`_IO uint8_t HAL_SD_CSDTypeDef::ManDefIECC`**  
Manufacturer default ECC
- **`_IO uint8_t HAL_SD_CSDTypeDef::WrSpeedFact`**  
Write speed factor
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxWrBlockLen`**  
Max. write data block length
- **`_IO uint8_t HAL_SD_CSDTypeDef::WriteBlockPaPartial`**  
Partial blocks for write allowed
- **`_IO uint8_t HAL_SD_CSDTypeDef::Reserved3`**  
Reserved
- **`_IO uint8_t HAL_SD_CSDTypeDef::ContentProtectAppli`**  
Content protection application
- **`_IO uint8_t HAL_SD_CSDTypeDef::FileFormatGrouop`**  
File format group

- `_IO uint8_t HAL_SD_CSDTypedef::CopyFlag`  
Copy flag (OTP)
- `_IO uint8_t HAL_SD_CSDTypedef::PermWrProtect`  
Permanent write protection
- `_IO uint8_t HAL_SD_CSDTypedef::TempWrProtect`  
Temporary write protection
- `_IO uint8_t HAL_SD_CSDTypedef::FileFormat`  
File format
- `_IO uint8_t HAL_SD_CSDTypedef::ECC`  
ECC code
- `_IO uint8_t HAL_SD_CSDTypedef::CSD_CRC`  
CSD CRC
- `_IO uint8_t HAL_SD_CSDTypedef::Reserved4`  
Always 1

### 38.1.3 HAL\_SD\_CIDTypedef

#### Data Fields

- `_IO uint8_t ManufacturerID`
- `_IO uint16_t OEM_AppId`
- `_IO uint32_t ProdName1`
- `_IO uint8_t ProdName2`
- `_IO uint8_t ProdRev`
- `_IO uint32_t ProdSN`
- `_IO uint8_t Reserved1`
- `_IO uint16_t ManufactDate`
- `_IO uint8_t CID_CRC`
- `_IO uint8_t Reserved2`

#### Field Documentation

- `_IO uint8_t HAL_SD_CIDTypedef::ManufacturerID`  
Manufacturer ID
- `_IO uint16_t HAL_SD_CIDTypedef::OEM_AppId`  
OEM/Application ID
- `_IO uint32_t HAL_SD_CIDTypedef::ProdName1`  
Product Name part1
- `_IO uint8_t HAL_SD_CIDTypedef::ProdName2`  
Product Name part2
- `_IO uint8_t HAL_SD_CIDTypedef::ProdRev`  
Product Revision
- `_IO uint32_t HAL_SD_CIDTypedef::ProdSN`  
Product Serial Number
- `_IO uint8_t HAL_SD_CIDTypedef::Reserved1`  
Reserved1
- `_IO uint16_t HAL_SD_CIDTypedef::ManufactDate`  
Manufacturing Date
- `_IO uint8_t HAL_SD_CIDTypedef::CID_CRC`  
CID CRC

- `_IO uint8_t HAL_SD_CID_TypeDef::Reserved2`  
Always 1

### 38.1.4 HAL\_SD\_CardStatusTypedef

#### Data Fields

- `_IO uint8_t DAT_BUS_WIDTH`
- `_IO uint8_t SECURED_MODE`
- `_IO uint16_t SD_CARD_TYPE`
- `_IO uint32_t SIZE_OF_PROTECTED_AREA`
- `_IO uint8_t SPEED_CLASS`
- `_IO uint8_t PERFORMANCE_MOVE`
- `_IO uint8_t AU_SIZE`
- `_IO uint16_t ERASE_SIZE`
- `_IO uint8_t ERASE_TIMEOUT`
- `_IO uint8_t ERASE_OFFSET`

#### Field Documentation

- `_IO uint8_t HAL_SD_CardStatusTypedef::DAT_BUS_WIDTH`  
Shows the currently defined data bus width
- `_IO uint8_t HAL_SD_CardStatusTypedef::SECURED_MODE`  
Card is in secured mode of operation
- `_IO uint16_t HAL_SD_CardStatusTypedef::SD_CARD_TYPE`  
Carries information about card type
- `_IO uint32_t HAL_SD_CardStatusTypedef::SIZE_OF_PROTECTED_AREA`  
Carries information about the capacity of protected area
- `_IO uint8_t HAL_SD_CardStatusTypedef::SPEED_CLASS`  
Carries information about the speed class of the card
- `_IO uint8_t HAL_SD_CardStatusTypedef::PERFORMANCE_MOVE`  
Carries information about the card's performance move
- `_IO uint8_t HAL_SD_CardStatusTypedef::AU_SIZE`  
Carries information about the card's allocation unit size
- `_IO uint16_t HAL_SD_CardStatusTypedef::ERASE_SIZE`  
Determines the number of AUs to be erased in one operation
- `_IO uint8_t HAL_SD_CardStatusTypedef::ERASE_TIMEOUT`  
Determines the timeout for any number of AU erase
- `_IO uint8_t HAL_SD_CardStatusTypedef::ERASE_OFFSET`  
Carries information about the erase offset

### 38.1.5 HAL\_SD\_CardInfoTypedef

#### Data Fields

- `HAL_SD_CSD_TypeDef SD_csd`
- `HAL_SD_CID_TypeDef SD_cid`
- `uint64_t CardCapacity`
- `uint32_t CardBlockSize`

- *uint16\_t RCA*
- *uint8\_t CardType*

#### Field Documentation

- *HAL\_SD\_CSDTypeDef HAL\_SD\_CardInfoTypedef::SD\_csd*  
SD card specific data register
- *HAL\_SD\_CIDTypeDef HAL\_SD\_CardInfoTypedef::SD\_cid*  
SD card identification number register
- *uint64\_t HAL\_SD\_CardInfoTypedef::CardCapacity*  
Card capacity
- *uint32\_t HAL\_SD\_CardInfoTypedef::CardBlockSize*  
Card block size
- *uint16\_t HAL\_SD\_CardInfoTypedef::RCA*  
SD relative card address
- *uint8\_t HAL\_SD\_CardInfoTypedef::CardType*  
SD card type

## 38.2 SD Firmware driver API description

### 38.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDIO and GPIO) are performed by the user in HAL\_SD\_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDIO memories which uses the HAL SDIO driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDIO low level resources by implement the HAL\_SD\_MspInit() API:
  - a. Enable the SDIO interface clock using \_\_HAL\_RCC\_SDIO\_CLK\_ENABLE();
  - b. SDIO pins configuration for SD card
    - Enable the clock for the SDIO GPIOs using the functions \_\_HAL\_RCC\_GPIOx\_CLK\_ENABLE();
    - Configure these SDIO pins as alternate function pull-up using HAL\_GPIO\_Init() and according to your pin assignment;
  - c. DMA Configuration if you need to use DMA process (HAL\_SD\_ReadBlocks\_DMA() and HAL\_SD\_WriteBlocks\_DMA() APIs).
    - Enable the DMAx interface clock using \_\_HAL\_RCC\_DMAx\_CLK\_ENABLE();
    - Configure the DMA using the function HAL\_DMA\_Init() with predeclared and filled.
  - d. NVIC configuration if you need to use interrupt process when using DMA transfer.
    - Configure the SDIO and DMA interrupt priorities using functions HAL\_NVIC\_SetPriority(); DMA priority is superior to SDIO's priority
    - Enable the NVIC DMA and SDIO IRQs using function HAL\_NVIC\_EnableIRQ()
    - SDIO interrupts are managed using the macros \_\_HAL\_SD\_SDIO\_ENABLE\_IT() and \_\_HAL\_SD\_SDIO\_DISABLE\_IT() inside the communication process.

- SDIO interrupts pending bits are managed using the macros  
  \_\_HAL\_SD\_SDIO\_GET\_IT() and \_\_HAL\_SD\_SDIO\_CLEAR\_IT()
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

## SD Card Initialization and configuration

To initialize the SD Card, use the HAL\_SD\_Init() function. It initializes the SD Card and put it into Standby State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDIO\_CK) is computed as follows: SDIO\_CK = SDIOCLK / (ClockDiv + 2) In initialization mode and according to the SD Card standard, make sure that the SDIO\_CK frequency doesn't exceed 400KHz.
2. Get the SD CID and CSD data. All these information are managed by the SDCardInfo structure. This structure provide also ready computed SD Card capacity and Block size. These information are stored in SD handle structure in case of future use.
3. Configure the SD Card Data transfer frequency. By Default, the card transfer frequency is set to 24MHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the SD Card standard, make sure that the SDIO\_CK frequency doesn't exceed 25MHz and 50MHz in High-speed mode switch. To be able to use a frequency higher than 24MHz, you should use the SDIO peripheral in bypass mode. Refer to the corresponding reference manual for more details.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

## SD Card Read operation

- You can read from SD card in polling mode by using function HAL\_SD\_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.
- You can read from SD card in DMA mode by using function HAL\_SD\_ReadBlocks\_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to call the function HAL\_SD\_CheckReadOperation(), to insure that the read transfer is done correctly in both DMA and SD sides.

## SD Card Write operation

- You can write to SD card in polling mode by using function HAL\_SD\_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.
- You can write to SD card in DMA mode by using function HAL\_SD\_WriteBlocks\_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks"

parameter. After this, you have to call the function HAL\_SD\_CheckWriteOperation(), to insure that the write transfer is done correctly in both DMA and SD sides.

### SD card status

- At any time, you can check the SD Card status and get the SD card state by using the HAL\_SD\_GetStatus() function. This function checks first if the SD card is still connected and then get the internal SD Card transfer state.
- You can also get the SD card SD Status register by using the HAL\_SD\_SendSDStatus() function.

### SD HAL driver macros list



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

#### 38.2.2

### Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

This section contains the following APIs:

- `HAL_SD_Init()`
- `HAL_SD_DelInit()`
- `HAL_SD_MspInit()`
- `HAL_SD_MspDelInit()`

#### 38.2.3

### IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

This section contains the following APIs:

- `HAL_SD_ReadBlocks()`
- `HAL_SD_WriteBlocks()`
- `HAL_SD_ReadBlocks_DMA()`
- `HAL_SD_WriteBlocks_DMA()`
- `HAL_SD_CheckReadOperation()`
- `HAL_SD_CheckWriteOperation()`
- `HAL_SD_Erase()`
- `HAL_SD_IRQHandler()`
- `HAL_SD_XferCpltCallback()`
- `HAL_SD_XferErrorCallback()`
- `HAL_SD_DMA_RxCpltCallback()`
- `HAL_SD_DMA_RxErrorCallback()`
- `HAL_SD_DMA_TxCpltCallback()`
- `HAL_SD_DMA_TxErrorCallback()`

#### 38.2.4

### Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations.

This section contains the following APIs:

- [\*HAL\\_SD\\_Get\\_CardInfo\(\)\*](#)
- [\*HAL\\_SD\\_WideBusOperation\\_Config\(\)\*](#)
- [\*HAL\\_SD\\_StopTransfer\(\)\*](#)
- [\*HAL\\_SD\\_HighSpeed\(\)\*](#)

### 38.2.5 Peripheral State functions

This subsection permits to get in runtime the status of the peripheral and the data flow.

This section contains the following APIs:

- [\*HAL\\_SD\\_SendSDStatus\(\)\*](#)
- [\*HAL\\_SD\\_GetStatus\(\)\*](#)
- [\*HAL\\_SD\\_GetCardStatus\(\)\*](#)

### 38.2.6 HAL\_SD\_Init

Function Name	<code>HAL_SD_ErrorTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * SDCardInfo)</code>
Function Description	Initializes the SD card according to the specified parameters in the <code>SD_HandleTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>SDCardInfo:</b> <code>HAL_SD_CardInfoTypeDef</code> structure for SD card information</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL SD error state</li> </ul>

### 38.2.7 HAL\_SD\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)</code>
Function Description	De-Initializes the SD card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 38.2.8 HAL\_SD\_MspInit

Function Name	<code>void HAL_SD_MspInit (SD_HandleTypeDef * hsd)</code>
Function Description	Initializes the SD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.9 HAL\_SD\_MspDeInit

Function Name	<code>void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)</code>
Function Description	De-Initialize SD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.10 HAL\_SD\_ReadBlocks

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</b>
Function Description	Reads block(s) from a specified address in a card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>pReadBuffer:</b> pointer to the buffer that will contain the received data</li> <li>• <b>ReadAddr:</b> Address from where data is to be read</li> <li>• <b>BlockSize:</b> SD card Data block size</li> <li>• <b>NumberOfBlocks:</b> Number of SD blocks to read</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• BlockSize must be 512 bytes.</li> </ul>

### 38.2.11 HAL\_SD\_WriteBlocks

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</b>
Function Description	Allows to write block(s) to a specified address in a card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>pWriteBuffer:</b> pointer to the buffer that will contain the data to transmit</li> <li>• <b>WriteAddr:</b> Address from where data is to be written</li> <li>• <b>BlockSize:</b> SD card Data block size</li> <li>• <b>NumberOfBlocks:</b> Number of SD blocks to write</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• BlockSize must be 512 bytes.</li> </ul>

### 38.2.12 HAL\_SD\_ReadBlocks\_DMA

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</b>
Function Description	Reads block(s) from a specified address in a card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>pReadBuffer:</b> Pointer to the buffer that will contain the received data</li> <li>• <b>ReadAddr:</b> Address from where data is to be read</li> <li>• <b>BlockSize:</b> SD card Data block size</li> <li>• <b>NumberOfBlocks:</b> Number of blocks to read.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API should be followed by the function HAL_SD_CheckReadOperation() to check the completion of the read process</li> <li>• BlockSize must be 512 bytes.</li> </ul>

### 38.2.13 HAL\_SD\_WriteBlocks\_DMA

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</b>
Function Description	Writes block(s) to a specified address in a card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>pWriteBuffer:</b> pointer to the buffer that will contain the data to transmit</li> <li>• <b>WriteAddr:</b> Address from where data is to be read</li> <li>• <b>BlockSize:</b> the SD card Data block size</li> <li>• <b>NumberOfBlocks:</b> Number of blocks to write</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This API should be followed by the function HAL_SD_CheckWriteOperation() to check the completion of the write process (by SD current status polling).</li> <li>• BlockSize must be 512 bytes.</li> </ul>

### 38.2.14 HAL\_SD\_CheckReadOperation

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_CheckReadOperation (SD_HandleTypeDef * hsd, uint32_t Timeout)</b>
Function Description	This function waits until the SD DMA data read transfer is finished.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 38.2.15 HAL\_SD\_CheckWriteOperation

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_CheckWriteOperation (SD_HandleTypeDef * hsd, uint32_t Timeout)</b>
Function Description	This function waits until the SD DMA data write transfer is finished.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 38.2.16 HAL\_SD\_Erase

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint64_t startaddr, uint64_t endaddr)</b>
Function Description	Erases the specified memory area of the given SD card.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>startaddr:</b> Start byte address</li> <li>• <b>endaddr:</b> End byte address</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 38.2.17 HAL\_SD\_IRQHandler

Function Name	<b>void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)</b>
Function Description	This function handles SD card interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.18 HAL\_SD\_XferCpltCallback

Function Name	<b>void HAL_SD_XferCpltCallback (SD_HandleTypeDef * hsd)</b>
Function Description	SD end of transfer callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.19 HAL\_SD\_XferErrorCallback

Function Name	<b>void HAL_SD_XferErrorCallback (SD_HandleTypeDef * hsd)</b>
Function Description	SD Transfer Error callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 38.2.20 HAL\_SD\_DMA\_RxCpltCallback

Function Name	<b>void HAL_SD_DMA_RxCpltCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	SD Transfer complete Rx callback in non blocking mode.
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>

### 38.2.21 HAL\_SD\_DMA\_RxErrorCallback

Function Name	<b>void HAL_SD_DMA_RxErrorCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	SD DMA transfer complete Rx error 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>

### 38.2.22 HAL\_SD\_DMA\_TxCpltCallback

Function Name	<b>void HAL_SD_DMA_TxCpltCallback (DMA_HandleTypeDef * hdma)</b>
Function Description	SD Transfer complete Tx callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma:</b> pointer to a DMA_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified DMA module.

Return values

- None

### 38.2.23 HAL\_SD\_DMA\_TxErrorCallback

Function Name      **void HAL\_SD\_DMA\_TxErrorCallback (DMA\_HandleTypeDef \* hdma)**

Function Description      SD DMA transfer complete error Tx callback.

Parameters

- **hdma:** pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- None

### 38.2.24 HAL\_SD\_Get\_CardInfo

Function Name      **HAL\_SD\_ErrorTypeDef HAL\_SD\_Get\_CardInfo (SD\_HandleTypeDef \* hsd, HAL\_SD\_CardInfoTypeDef \* pCardInfo)**

Function Description      Returns information about specific card.

Parameters

- **hsd:** SD handle
- **pCardInfo:** Pointer to a HAL\_SD\_CardInfoTypeDef structure that contains all SD card information

Return values

- SD Card error state

### 38.2.25 HAL\_SD\_WideBusOperation\_Config

Function Name      **HAL\_SD\_ErrorTypeDef HAL\_SD\_WideBusOperation\_Config (SD\_HandleTypeDef \* hsd, uint32\_t WideMode)**

Function Description      Enables wide bus operation for the requested card if supported by card.

Parameters

- **hsd:** SD handle
- **WideMode:** Specifies the SD card wide bus mode This parameter can be one of the following values:  
SDIO\_BUS\_WIDE\_8B: 8-bit data transfer (Only for MMC)  
SDIO\_BUS\_WIDE\_4B: 4-bit data transfer  
SDIO\_BUS\_WIDE\_1B: 1-bit data transfer

Return values

- SD Card error state

### 38.2.26 HAL\_SD\_StopTransfer

Function Name      **HAL\_SD\_ErrorTypeDef HAL\_SD\_StopTransfer (SD\_HandleTypeDef \* hsd)**

Function Description      Aborts an ongoing data transfer.

Parameters

- **hsd:** SD handle

Return values

- SD Card error state

### 38.2.27 HAL\_SD\_HighSpeed

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_HighSpeed (SD_HandleTypeDef * hsd)</b>
Function Description	Switches the SD card to High Speed mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This operation should be followed by the configuration of PLL to have SDIOCK clock between 67 and 75 MHz</li> </ul>

### 38.2.28 HAL\_SD\_SendSDStatus

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_SendSDStatus (SD_HandleTypeDef * hsd, uint32_t * pSDstatus)</b>
Function Description	Returns the current SD card's status.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>pSDstatus:</b> Pointer to the buffer that will contain the SD card status SD Status register)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

### 38.2.29 HAL\_SD\_GetStatus

Function Name	<b>HAL_SD_TransferStateTypeDef HAL_SD_GetStatus (SD_HandleTypeDef * hsd)</b>
Function Description	Gets the current sd card data status.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Data Transfer state</li> </ul>

### 38.2.30 HAL\_SD\_GetCardStatus

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pCardStatus)</b>
Function Description	Gets the SD card status.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd:</b> SD handle</li> <li>• <b>pCardStatus:</b> Pointer to the HAL_SD_CardStatusTypeDef structure that will contain the SD card status information</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• SD Card error state</li> </ul>

## 38.3 SD Firmware driver defines

### 38.3.1 SD

#### *SD Exported Constants*

SD_CMD_GO_IDLE_STATE	Resets the SD memory card.
----------------------	----------------------------

SD_CMD_SEND_OP_COND	Sends host capacity support information and activates the card's initialization process.
SD_CMD_ALL_SEND_CID	Asks any card connected to the host to send the CID numbers on the CMD line.
SD_CMD_SET_REL_ADDR	Asks the card to publish a new relative address (RCA).
SD_CMD_SET_DSR	Programs the DSR of all cards.
SD_CMD_SDIO_SEN_OP_COND	Sends host capacity support information (HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_HS_SWITCH	Checks switchable function (mode 0) and switch card function (mode 1).
SD_CMD_SEL_DESEL_CARD	Selects the card by its own relative address and gets deselected by any other address
SD_CMD_HS_SEND_EXT_CSD	Sends SD Memory Card interface condition, which includes host supply voltage information and asks the card whether card supports voltage.
SD_CMD_SEND_CSD	Addressed card sends its card specific data (CSD) on the CMD line.
SD_CMD_SEND_CID	Addressed card sends its card identification (CID) on the CMD line.
SD_CMD_READ_DAT_UNTIL_STOP	SD card doesn't support it.
SD_CMD_STOP_TRANSMISSION	Forces the card to stop transmission.
SD_CMD_SEND_STATUS	Addressed card sends its status register.
SD_CMD_HS_BUSTEST_READ	
SD_CMD_GO_INACTIVE_STATE	Sends an addressed card into the inactive state.
SD_CMD_SET_BLOCKLEN	Sets the block length (in bytes for SDSC) for all following block commands

	(read, write, lock). Default block length is fixed to 512 Bytes. Not effective for SDHS and SDXC.
SD_CMD_READ_SINGLE_BLOCK	Reads single block of size selected by SET_BLOCKLEN in case of SDSC, and a block of fixed 512 bytes in case of SDHC and SDXC.
SD_CMD_READ_MULT_BLOCK	Continuously transfers data blocks from card to host until interrupted by STOP_TRANSMISSION command.
SD_CMD_HS_BUSTEST_WRITE	64 bytes tuning pattern is sent for SDR50 and SDR104.
SD_CMD_WRITE_DAT_UNTIL_STOP	Speed class control command.
SD_CMD_SET_BLOCK_COUNT	Specify block count for CMD18 and CMD25.
SD_CMD_WRITE_SINGLE_BLOCK	Writes single block of size selected by SET_BLOCKLEN in case of SDSC, and a block of fixed 512 bytes in case of SDHC and SDXC.
SD_CMD_WRITE_MULT_BLOCK	Continuously writes blocks of data until a STOP_TRANSMISSION follows.
SD_CMD_PROG_CID	Reserved for manufacturers.
SD_CMD_PROG_CSD	Programming of the programmable bits of the CSD.
SD_CMD_SET_WRITE_PROT	Sets the write protection bit of the addressed group.
SD_CMD_CLR_WRITE_PROT	Clears the write protection bit of the addressed group.
SD_CMD_SEND_WRITE_PROT	Asks the card to send the status of the write protection bits.
SD_CMD_SD_ERASE_GRP_START	Sets the address of the first write block to be erased. (For SD card only).
SD_CMD_SD_ERASE_GRP_END	Sets the address of the last write block of the continuous

SD_CMD_ERASE_GRP_START	range to be erased.
SD_CMD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE	Sets the address of the first write block to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_FAST_IO	Reserved for SD security applications.
SD_CMD_GO_IRQ_STATE	Reserved for each command system set by switch function command (CMD6).
SD_CMD_LOCK_UNLOCK	SD card doesn't support it (Reserved).
SD_CMD_APP_CMD	SD card doesn't support it (Reserved).
SD_CMD_GEN_CMD	Sets/resets the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
SD_CMD_NO_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
SD_CMD_APP_SD_SET_BUSWIDTH	Used either to transfer a data block to the card or to get a data block from the card for general purpose/application specific commands.
SD_CMD_SD_APP_STATUS	SDIO_APP_CMD should be sent before sending these commands. (ACMD6)
SD_CMD_SD_APP_SEND_NUM_WRITE_BLOCKS	Defines the data bus width to be used for data transfer. The allowed data bus widths are given in SCR register. (ACMD13) Sends the SD status. (ACMD22) Sends the number of the written (without errors) write blocks.

	Responds with 32bit+CRC data block.
SD_CMD_SD_APP_OP_COND	(ACMD41) Sends host capacity support information (HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_SD_APP_SET_CLR_CARD_DETECT	(ACMD42) Connects/Disconnects the 50 KOhm pull-up resistor on CD/DAT3 (pin 1) of the card.
SD_CMD_SD_APP_SEND_SCR	Reads the SD Configuration Register (SCR).
SD_CMD_SDIO_RW_DIRECT	For SD I/O card only, reserved for security specification.
SD_CMD_SDIO_RW_EXTENDED	For SD I/O card only, reserved for security specification.
SD_CMD_SD_APP_GET_MKB	SD_CMD_APP_CMD should be sent before sending these commands. For SD card only
SD_CMD_SD_APP_GET_MID	For SD card only
SD_CMD_SD_APP_SET_CER_RN1	For SD card only
SD_CMD_SD_APP_GET_CER_RN2	For SD card only
SD_CMD_SD_APP_SET_CER_RES2	For SD card only
SD_CMD_SD_APP_GET_CER_RES1	For SD card only
SD_CMD_SD_APP_SECURE_READ_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_ERASE	For SD card only
SD_CMD_SD_APP_CHANGE_SECURE_AREA	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MKB	For SD card only
STD_CAPACITY_SD_CARD_V1_1	
STD_CAPACITY_SD_CARD_V2_0	
HIGH_CAPACITY_SD_CARD	
MULTIMEDIA_CARD	
SECURE_DIGITAL_IO_CARD	
HIGH_SPEED_MULTIMEDIA_CARD	
SECURE_DIGITAL_IO_COMBO_CARD	
HIGH_CAPACITY_MMICARD	

***SD Exported Macros*****`_HAL_SD_SDIO_ENABLE`****Description:**

- Enable the SD device.

**Return value:**

- None

**`_HAL_SD_SDIO_DISABLE`****Description:**

- Disable the SD device.

**Return value:**

- None

**`_HAL_SD_SDIO_DMA_ENABLE`****Description:**

- Enable the SDIO DMA transfer.

**Return value:**

- None

**`_HAL_SD_SDIO_DMA_DISABLE`****Description:**

- Disable the SDIO DMA transfer.

**Return value:**

- None

**`_HAL_SD_SDIO_ENABLE_IT`****Description:**

- Enable the SD device interrupt.

**Parameters:**

- `_HANDLE_`: SD Handle
- `_INTERRUPT_`: specifies the SDIO interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
  - `SDIO_IT_DCRCFAIL`: Data block sent/received (CRC check failed) interrupt
  - `SDIO_IT_CTIMEOUT`: Command response timeout interrupt
  - `SDIO_IT_DTIMEOUT`: Data timeout interrupt
  - `SDIO_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
  - `SDIO_IT_RXOVERR`: Received FIFO overrun error interrupt
  - `SDIO_IT_CMDREND`: Command response received (CRC check passed) interrupt
  - `SDIO_IT_CMDSENT`: Command sent (no response required) interrupt
  - `SDIO_IT_DATAEND`: Data end (data counter, `SDIDCOUNT`, is zero) interrupt
  - `SDIO_IT_STBITERR`: Start bit not detected on all data signals in wide bus mode

- interrupt
- SDIO\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDIO\_IT\_CMDACT: Command transfer in progress interrupt
  - SDIO\_IT\_TXACT: Data transmit in progress interrupt
  - SDIO\_IT\_RXACT: Data receive in progress interrupt
  - SDIO\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDIO\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDIO\_IT\_TXFIFOF: Transmit FIFO full interrupt
  - SDIO\_IT\_RXFIFOF: Receive FIFO full interrupt
  - SDIO\_IT\_TXFIFOE: Transmit FIFO empty interrupt
  - SDIO\_IT\_RXFIFOE: Receive FIFO empty interrupt
  - SDIO\_IT\_TXDAVL: Data available in transmit FIFO interrupt
  - SDIO\_IT\_RXDAVL: Data available in receive FIFO interrupt
  - SDIO\_IT\_SDIOIT: SD I/O interrupt received interrupt
  - SDIO\_IT\_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

**Return value:**

- None

**\_\_HAL\_SD\_SDIO\_DISABLE\_IT****Description:**

- Disable the SD device interrupt.

**Parameters:**

- \_\_HANDLE\_\_: SD Handle
- \_\_INTERRUPT\_\_: specifies the SDIO interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDIO\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO\_IT\_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO\_IT\_CTIMEOUT: Command response timeout interrupt
  - SDIO\_IT\_DTIMEOUT: Data timeout interrupt
  - SDIO\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDIO\_IT\_RXOVERR: Received FIFO overrun error interrupt

- SDIO\_IT\_CMDREND: Command response received (CRC check passed) interrupt
- SDIO\_IT\_CMDSENT: Command sent (no response required) interrupt
- SDIO\_IT\_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO\_IT\_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO\_IT\_CMDACT: Command transfer in progress interrupt
- SDIO\_IT\_TXACT: Data transmit in progress interrupt
- SDIO\_IT\_RXACT: Data receive in progress interrupt
- SDIO\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO\_IT\_TXFIFOF: Transmit FIFO full interrupt
- SDIO\_IT\_RXFIFOF: Receive FIFO full interrupt
- SDIO\_IT\_TXFIFOE: Transmit FIFO empty interrupt
- SDIO\_IT\_RXFIFOE: Receive FIFO empty interrupt
- SDIO\_IT\_TXDAVL: Data available in transmit FIFO interrupt
- SDIO\_IT\_RXDAVL: Data available in receive FIFO interrupt
- SDIO\_IT\_SDIOIT: SD I/O interrupt received interrupt
- SDIO\_IT\_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

**Return value:**

- None

**\_HAL\_SD\_SDIO\_GET\_FLAG****Description:**

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

**Parameters:**

- \_HANDLE\_: SD Handle
- \_FLAG\_: specifies the flag to check. This parameter can be one of the following values:
  - SDIO\_FLAG\_CCRCFAIL: Command response received (CRC check failed)
  - SDIO\_FLAG\_DCRCFAIL: Data block sent/received (CRC check failed)

- SDIO\_FLAG\_CTIMEOUT: Command response timeout
- SDIO\_FLAG\_DTIMEOUT: Data timeout
- SDIO\_FLAG\_TXUNDERR: Transmit FIFO underrun error
- SDIO\_FLAG\_RXOVERR: Received FIFO overrun error
- SDIO\_FLAG\_CMDREND: Command response received (CRC check passed)
- SDIO\_FLAG\_CMDSENT: Command sent (no response required)
- SDIO\_FLAG\_DATAEND: Data end (data counter, SDIDCOUNT, is zero)
- SDIO\_FLAG\_STBITERR: Start bit not detected on all data signals in wide bus mode.
- SDIO\_FLAG\_DBCKEND: Data block sent/received (CRC check passed)
- SDIO\_FLAG\_CMDACT: Command transfer in progress
- SDIO\_FLAG\_TXACT: Data transmit in progress
- SDIO\_FLAG\_RXACT: Data receive in progress
- SDIO\_FLAG\_TXFIFOHE: Transmit FIFO Half Empty
- SDIO\_FLAG\_RXFIFOHF: Receive FIFO Half Full
- SDIO\_FLAG\_TXFIFOF: Transmit FIFO full
- SDIO\_FLAG\_RXFIFOF: Receive FIFO full
- SDIO\_FLAG\_TXFIFOE: Transmit FIFO empty
- SDIO\_FLAG\_RXFIFOE: Receive FIFO empty
- SDIO\_FLAG\_TXDAVL: Data available in transmit FIFO
- SDIO\_FLAG\_RXDAVL: Data available in receive FIFO
- SDIO\_FLAG\_SDIOIT: SD I/O interrupt received
- SDIO\_FLAG\_CEATAEND: CE-ATA command completion signal received for CMD61

**Return value:**

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

[\\_\\_HAL\\_SD\\_SDIO\\_CLEAR\\_FLAG](#)**Description:**

- Clear the SD's pending flags.

**Parameters:**

- \_\_HANDLE\_\_: SD Handle
- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be one or a combination of the

following values:

- SDIO\_FLAG\_CCRCFAIL: Command response received (CRC check failed)
- SDIO\_FLAG\_DCRCFAIL: Data block sent/received (CRC check failed)
- SDIO\_FLAG\_CTIMEOUT: Command response timeout
- SDIO\_FLAG\_DTIMEOUT: Data timeout
- SDIO\_FLAG\_TXUNDERR: Transmit FIFO underrun error
- SDIO\_FLAG\_RXOVERR: Received FIFO overrun error
- SDIO\_FLAG\_CMDREND: Command response received (CRC check passed)
- SDIO\_FLAG\_CMDSENT: Command sent (no response required)
- SDIO\_FLAG\_DATAEND: Data end (data counter, SDIDCOUNT, is zero)
- SDIO\_FLAG\_STBITERR: Start bit not detected on all data signals in wide bus mode
- SDIO\_FLAG\_DBCKEND: Data block sent/received (CRC check passed)
- SDIO\_FLAG\_SDIOIT: SD I/O interrupt received
- SDIO\_FLAG\_CEATAEND: CE-ATA command completion signal received for CMD61

#### **Return value:**

- None

### **\_\_HAL\_SD\_SDIO\_GET\_IT**

- Check whether the specified SD interrupt has occurred or not.

#### **Parameters:**

- **\_\_HANDLE\_\_**: SD Handle
- **\_\_INTERRUPT\_\_**: specifies the SDIO interrupt source to check. This parameter can be one of the following values:
  - SDIO\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO\_IT\_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO\_IT\_CTIMEOUT: Command response timeout interrupt
  - SDIO\_IT\_DTIMEOUT: Data timeout interrupt
  - SDIO\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDIO\_IT\_RXOVERR: Received FIFO overrun error interrupt
  - SDIO\_IT\_CMDREND: Command response

- received (CRC check passed) interrupt
- SDIO\_IT\_CMDSENT: Command sent (no response required) interrupt
- SDIO\_IT\_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO\_IT\_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO\_IT\_CMDACT: Command transfer in progress interrupt
- SDIO\_IT\_TXACT: Data transmit in progress interrupt
- SDIO\_IT\_RXACT: Data receive in progress interrupt
- SDIO\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO\_IT\_TXFIFOF: Transmit FIFO full interrupt
- SDIO\_IT\_RXFIFOF: Receive FIFO full interrupt
- SDIO\_IT\_TXFIFOE: Transmit FIFO empty interrupt
- SDIO\_IT\_RXFIFOE: Receive FIFO empty interrupt
- SDIO\_IT\_TXDABL: Data available in transmit FIFO interrupt
- SDIO\_IT\_RXDABL: Data available in receive FIFO interrupt
- SDIO\_IT\_SDIOIT: SD I/O interrupt received interrupt
- SDIO\_IT\_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

**Return value:**

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

[\\_\\_HAL\\_SD\\_SDIO\\_CLEAR\\_IT](#)**Description:**

- Clear the SD's interrupt pending bits.

**Parameters:**

- \_\_HANDLE\_\_: : SD Handle
- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDIO\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDIO\_IT\_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDIO\_IT\_CTIMEOUT: Command response

- timeout interrupt
- SDIO\_IT\_DTIMEOUT: Data timeout interrupt
- SDIO\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
- SDIO\_IT\_RXOVERR: Received FIFO overrun error interrupt
- SDIO\_IT\_CMDREND: Command response received (CRC check passed) interrupt
- SDIO\_IT\_CMDSENT: Command sent (no response required) interrupt
- SDIO\_IT\_DATAEND: Data end (data counter, SDIO\_DCOUNT, is zero) interrupt
- SDIO\_IT\_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO\_IT\_SDIOIT: SD I/O interrupt received interrupt
- SDIO\_IT\_CEATAEND: CE-ATA command completion signal received for CMD61

**Return value:**

- None

***SD Handle Structure definition***`SD_InitTypeDef``SD_TypeDef`***SD Private Defines***`DATA_BLOCK_SIZE``SDIO_STATIC_FLAGS``SDIO_CMD0TIMEOUT``SD_OCR_ADDR_OUT_OF_RANGE``SD_OCR_ADDR_MISALIGNED``SD_OCR_BLOCK_LEN_ERR``SD_OCR_ERASE_SEQ_ERR``SD_OCR_BAD_ERASE_PARAM``SD_OCR_WRITE_PROT_VIOLATION``SD_OCR_LOCK_UNLOCK_FAILED``SD_OCR_COM_CRC_FAILED``SD_OCR_ILLEGAL_CMD``SD_OCR_CARD_ECC_FAILED``SD_OCR_CC_ERROR``SD_OCR_GENERAL_UNKNOWN_ERROR``SD_OCR_STREAM_READ_UNDERRUN`

SD_OCR_STREAM_WRITE_OVERRUN	
SD_OCR_CID_CSD_OVERWRITE	
SD_OCR_WP_ERASE_SKIP	
SD_OCR_CARD_ECC_DISABLED	
SD_OCR_ERASE_RESET	
SD_OCR_AKE_SEQ_ERROR	
SD_OCR_ERRORBITS	
SD_R6_GENERAL_UNKNOWN_ERROR	
SD_R6_ILLEGAL_CMD	
SD_R6_COM_CRC_FAILED	
SD_VOLTAGE_WINDOW_SD	
SD_HIGH_CAPACITY	
SD_STD_CAPACITY	
SD_CHECK_PATTERN	
SD_MAX_VOLT_TRIAL	
SD_ALLZERO	
SD_WIDE_BUS_SUPPORT	
SD_SINGLE_BUS_SUPPORT	
SD_CARD_LOCKED	
SD_DATATIMEOUT	
SD_0TO7BITS	
SD_8TO15BITS	
SD_16TO23BITS	
SD_24TO31BITS	
SD_MAX_DATA_LENGTH	
SD_HALFFIFO	
SD_HALFFIFOBYTES	
SD_CCCC_LOCK_UNLOCK	
SD_CCCC_WRITE_PROT	
SD_CCCC_ERASE	
SD_SDIO_SEND_IF_COND	SDIO_APP_CMD should be sent before sending these commands.

## 39 HAL SMARTCARD Generic Driver

### 39.1 SMARTCARD Firmware driver registers structures

#### 39.1.1 SMARTCARD\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t CLKLastBit*
- *uint32\_t Prescaler*
- *uint32\_t 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:  

$$\text{IntegerDivider} = ((\text{PCLKx}) / (8 * (\text{hirda} > \text{Init.BaudRate})))$$

$$\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{uint32_t}) \text{ IntegerDivider})) * 8) + 0.5$$
- ***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\*\*](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32\_t SMARTCARD\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [\*\*SMARTCARD\\_Mode\*\*](#)
- ***uint32\_t SMARTCARD\_InitTypeDef::CLKPolarity***  
Specifies the steady state of the serial clock. This parameter can be a value of [\*\*SMARTCARD\\_Clock\\_Polarity\*\*](#)
- ***uint32\_t SMARTCARD\_InitTypeDef::CLKPhase***  
Specifies the clock transition on which the bit capture is made. This parameter can be a value of [\*\*SMARTCARD\\_Clock\\_Phase\*\*](#)
- ***uint32\_t SMARTCARD\_InitTypeDef::CLKLastBit***  
Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB)

- has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD\_Last\_Bit***
- ***uint32\_t SMARTCARD\_InitTypeDef::Prescaler***  
Specifies the SmartCard Prescaler value used for dividing the system clock to provide the smartcard clock. This parameter can be a value of ***SMARTCARD\_Prescaler***
  - ***uint32\_t SMARTCARD\_InitTypeDef::GuardTime***  
Specifies the SmartCard Guard Time value in terms of number of baud clocks. The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency
  - ***uint32\_t SMARTCARD\_InitTypeDef::NACKState***  
Specifies the SmartCard NACK Transmission state. This parameter can be a value of ***SMARTCARD\_NACK\_State***

### 39.1.2 SMARTCARD\_HandleTypeDef

#### Data Fields

- ***USART\_TypeDef \* Instance***
- ***SMARTCARD\_InitTypeDef Init***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferSize***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_SMARTCARD\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***USART\_TypeDef\* SMARTCARD\_HandleTypeDef::Instance***
- ***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 uint32\_t SMARTCARD\_HandleTypeDef::ErrorCode***

## 39.2 SMARTCARD Firmware driver API description

### 39.2.1 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 \_\_HAL\_SMARTCARD\_ENABLE\_IT() and \_\_HAL\_SMARTCARD\_DISABLE\_IT() inside the transmit and receive process.

Three operation modes are available within this driver :

#### Polling mode IO operation

- Send an amount of data in blocking mode using HAL\_SMARTCARD\_Transmit()
- Receive an amount of data in blocking mode using HAL\_SMARTCARD\_Receive()

#### Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL\_SMARTCARD\_Transmit\_IT()

- At transmission end of transfer HAL\_SMARTCARD\_TxCpltCallback is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_TxCpltCallback
- Receive an amount of data in non blocking mode using  
HAL\_SMARTCARD\_Receive\_IT()
- At reception end of transfer HAL\_SMARTCARD\_RxCpltCallback is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_RxCpltCallback
- In case of transfer Error, HAL\_SMARTCARD\_ErrorCallback() function is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using  
HAL\_SMARTCARD\_Transmit\_DMA()
- At transmission end of transfer HAL\_SMARTCARD\_TxCpltCallback is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using  
HAL\_SMARTCARD\_Receive\_DMA()
- At reception end of transfer HAL\_SMARTCARD\_RxCpltCallback is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_RxCpltCallback
- In case of transfer Error, HAL\_SMARTCARD\_ErrorCallback() function is executed and user can add his own code by customization of function pointer  
HAL\_SMARTCARD\_ErrorCallback

### SMARTCARD HAL driver macros list

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

- \_\_HAL\_SMARTCARD\_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

#### 39.2.2

#### Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in Smartcard mode.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

- For the Smartcard mode only these parameters can be configured:
  - Baud Rate
  - Word Length => Should be 9 bits (8 bits + parity)
  - Stop Bit
  - Parity: => Should be enabled (see table below)
  - USART polarity
  - USART phase
  - USART LastBit
  - Receiver/transmitter modes
  - Prescaler
  - GuardTime
  - NACKState: The Smartcard NACK state
- Recommended SmartCard interface configuration to get the Answer to Reset from the Card:
  - Word Length = 9 Bits
  - 1.5 Stop Bit
  - Even parity
  - BaudRate = 12096 baud
  - Tx and Rx enabled

**Table 17: USART frame formats**

M bit	PCE bit	Smartcard frame
1	1	SB   8 bit data   PB   STB

Please refer to the ISO 7816-3 specification for more details. -@- It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.

The HAL\_SMARTCARD\_Init() function follows the USART SmartCard configuration procedure (details for the procedure are available in reference manual (RM0329)).

This section contains the following APIs:

- [\*\*HAL\\_SMARTCARD\\_Init\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_MspDeInit\(\)\*\*](#)
- [\*\*HAL\\_SMARTCARD\\_Reliit\(\)\*\*](#)

### 39.2.3

### IO operation functions

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as: (+) 8 bits plus parity: where M=1 and PCE=1 in the USART\_CR1 register (+) 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART\_CR2 register.

1. There are two modes of transfer:

- Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - Non Blocking mode: The communication is performed using Interrupts or DMA, These 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()

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART\_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART\_CR2 register. (#) There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - Non Blocking mode: The communication is performed using Interrupts or DMA, 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 (#) Blocking mode APIs are :
    - HAL\_SMARTCARD\_Transmit()
    - HAL\_SMARTCARD\_Receive() (#) Non Blocking mode APIs with Interrupt are :
      - HAL\_SMARTCARD\_Transmit\_IT()
      - HAL\_SMARTCARD\_Receive\_IT()
      - HAL\_SMARTCARD\_IRQHandler() (#) Non Blocking mode functions with DMA are :
        - HAL\_SMARTCARD\_Transmit\_DMA()
        - HAL\_SMARTCARD\_Receive\_DMA() (#) A set of Transfer Complete Callbacks are provided in non Blocking mode:
          - HAL\_SMARTCARD\_TxCpltCallback()
          - HAL\_SMARTCARD\_RxCpltCallback()

- HAL\_SMARTCARD\_ErrorCallback()

This section contains the following APIs:

- [\*\*\*HAL\\_SMARTCARD\\_Transmit\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_Receive\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_Transmit\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_Receive\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_Transmit\\_DMA\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_Receive\\_DMA\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_IRQHandler\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_TxCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_RxCpltCallback\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_ErrorCallback\(\)\*\*\*](#)

### 39.2.4 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.

This section contains the following APIs:

- [\*\*\*HAL\\_SMARTCARD\\_GetState\(\)\*\*\*](#)
- [\*\*\*HAL\\_SMARTCARD\\_GetError\(\)\*\*\*](#)

### 39.2.5 HAL\_SMARTCARD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Init(SMARTCARD_HandleTypeDef * hsc)</b>
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>• HAL status</li> </ul>

### 39.2.6 HAL\_SMARTCARD\_DelInit

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_DelInit(SMARTCARD_HandleTypeDef * hsc)</b>
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>• HAL status</li> </ul>

### 39.2.7 HAL\_SMARTCARD\_MspInit

Function Name	<b>void HAL_SMARTCARD_MspInit</b>
---------------	-----------------------------------

**(SMARTCARD\_HandleTypeDef \* hsc)**

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>

### 39.2.8 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>

### 39.2.9 HAL\_SMARTCARD\_RelInit

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_RelInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	

### 39.2.10 HAL\_SMARTCARD\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 39.2.11 HAL\_SMARTCARD\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsc, 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>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>

- **Timeout:** Timeout duration

Return values • HAL status

### 39.2.12 HAL\_SMARTCARD\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsc, 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>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	• HAL status

### 39.2.13 HAL\_SMARTCARD\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t 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	• HAL status

### 39.2.14 HAL\_SMARTCARD\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsc, 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>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	• HAL status

### 39.2.15 HAL\_SMARTCARD\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t 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>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When the SMARTCARD parity is enabled (PCE = 1) the data received contain the parity bits.</li> </ul>

### 39.2.16 HAL\_SMARTCARD\_IRQHandler

Function Name	<b>void HAL_SMARTCARD_IRQHandler(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>

### 39.2.17 HAL\_SMARTCARD\_TxCpltCallback

Function Name	<b>void HAL_SMARTCARD_TxCpltCallback(SMARTCARD_HandleTypeDef * 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>

### 39.2.18 HAL\_SMARTCARD\_RxCpltCallback

Function Name	<b>void HAL_SMARTCARD_RxCpltCallback(SMARTCARD_HandleTypeDef * 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>

### 39.2.19 HAL\_SMARTCARD\_ErrorCallback

Function Name	<b>void HAL_SMARTCARD_ErrorCallback(SMARTCARD_HandleTypeDef * 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>
---------------	--

### 39.2.20 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>• HAL state</li> </ul>

### 39.2.21 HAL\_SMARTCARD\_GetError

Function Name	<b>uint32_t HAL_SMARTCARD_GetError</b> <b>(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>• SMARTCARD Error Code</li> </ul>

## 39.3 SMARTCARD Firmware driver defines

### 39.3.1 SMARTCARD

***SMARTCARD Clock Phase***

SMARTCARD\_PHASE\_1EDGE

SMARTCARD\_PHASE\_2EDGE

***SMARTCARD Clock Polarity***

SMARTCARD\_POLARITY\_LOW

SMARTCARD\_POLARITY\_HIGH

***SMARTCARD DMA requests***

SMARTCARD\_DMAREQ\_TX

SMARTCARD\_DMAREQ\_RX

***SMARTCARD Error Code***

HAL_SMARTCARD_ERROR_NONE	No error
--------------------------	----------

HAL_SMARTCARD_ERROR_PE	Parity error
------------------------	--------------

HAL_SMARTCARD_ERROR_NE	Noise error
------------------------	-------------

HAL_SMARTCARD_ERROR_FE	Frame error
------------------------	-------------

HAL_SMARTCARD_ERROR_ORE	Overrun error
-------------------------	---------------

HAL_SMARTCARD_ERROR_DMA	DMA transfer error
-------------------------	--------------------

***SMARTCARD Exported Macros*****`_HAL_SMARTCARD_RESET_HANDLE_STA  
TE`****Description:**

- Reset SMARTCARD handle state.

**Parameters:**

- `_HANDLE_`: specifies the SMARTCARD Handle.

**Return value:**

- None

**`_HAL_SMARTCARD_FLUSH_DRREGISTER`****Description:**

- Flushes the Smartcard DR register.

**Parameters:**

- `_HANDLE_`: specifies the SMARTCARD Handle.

**`_HAL_SMARTCARD_GET_FLAG`****Description:**

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

**Parameters:**

- `_HANDLE_`: specifies the SMARTCARD Handle.
- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - `SMARTCARD_FLAG_TXE`: Transmit data register empty flag
  - `SMARTCARD_FLAG_TC`: Transmission Complete flag
  - `SMARTCARD_FLAG_RXNE`: Receive data register not empty flag
  - `SMARTCARD_FLAG_IDLE`: Idle Line detection flag
  - `SMARTCARD_FLAG_ORE`: Overrun Error flag
  - `SMARTCARD_FLAG_NE`: Noise Error flag
  - `SMARTCARD_FLAG_FE`: Framing Error flag
  - `SMARTCARD_FLAG_PE`: Parity Error flag

**Return value:**

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

**`_HAL_SMARTCARD_CLEAR_FLAG`****Description:**

- Clears the specified Smartcard pending flags.

**Parameters:**

- HANDLE: specifies the SMARTCARD Handle.
- FLAG: specifies the flag to check. This parameter can be any combination of the following values:
  - SMARTCARD\_FLAG\_TC: Transmission Complete flag.
  - SMARTCARD\_FLAG\_RXNE: Receive data register not empty flag.

**Notes:**

- PE (Parity error), FE (Framing error), NE (Noise error) and ORE (Overrun error) flags are cleared by software sequence: a read operation to USART\_SR register followed by a read operation to USART\_DR register. RXNE flag can be also cleared by a read to the USART\_DR register. TC flag can be also cleared by software sequence: a read operation to USART\_SR register followed by a write operation to USART\_DR register. TXE flag is cleared only by a write to the USART\_DR register.

\_\_HAL\_SMARTCARD\_CLEAR\_PEFLAG

**Description:**

- Clear the SMARTCARD PE pending flag.

**Parameters:**

- HANDLE: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

\_\_HAL\_SMARTCARD\_CLEAR\_FEFLAG

**Description:**

- Clear the SMARTCARD FE pending flag.

**Parameters:**

- HANDLE: specifies the USART Handle. This parameter

can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_SMARTCARD_CLEAR_NEFLAG`

**Description:**

- Clear the SMARTCARD NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_SMARTCARD_CLEAR_OREFLAG`

**Description:**

- Clear the SMARTCARD ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_SMARTCARD_CLEAR_IDLEFLAG`

**Description:**

- Clear the SMARTCARD IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_SMARTCARD_ENABLE_IT`

**Description:**

- Enables or disables the specified SmartCard interrupts.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle.
- \_\_INTERRUPT\_\_: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_IDLE: Idle line detection interrupt
  - SMARTCARD\_IT\_PE: Parity Error interrupt
  - SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

\_\_HAL\_SMARTCARD\_DISABLE\_IT  
\_\_HAL\_SMARTCARD\_GET\_IT\_SOURCE

#### Description:

- Checks whether the specified SmartCard interrupt has occurred or not.

#### Parameters:

- \_\_HANDLE\_\_: specifies the SmartCard Handle.
- \_\_IT\_\_: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_IDLE: Idle line detection interrupt
  - SMARTCARD\_IT\_ERR: Error interrupt
  - SMARTCARD\_IT\_PE: Parity Error interrupt

#### Return value:

- The: new state of \_\_IT\_\_ (TRUE or

FALSE).

`__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE`

**Description:**

- Macro to enable the SMARTCARD's one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle.

**Return value:**

- None

`__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE`

**Description:**

- Macro to disable the SMARTCARD's one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle.

**Return value:**

- None

`__HAL_SMARTCARD_ENABLE`

**Description:**

- Enable the USART associated to the SMARTCARD Handle.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

`__HAL_SMARTCARD_DISABLE`

**Description:**

- Disable the USART associated to the SMARTCARD Handle.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

`_HAL_SMARTCARD_DMA_REQUEST_ENAB  
LE`

**Description:**

- Macros to enable or disable the SmartCard DMA request.

**Parameters:**

- `_HANDLE_`: specifies the SmartCard Handle.
- `_REQUEST_`: specifies the SmartCard DMA request. This parameter can be one of the following values:
  - `SMARTCARD_DMAREQ_TX`: SmartCard DMA transmit request
  - `SMARTCARD_DMAREQ_RX`: SmartCard DMA receive request

`_HAL_SMARTCARD_DMA_REQUEST_DISA  
BLE`

**SMARTCARD Flags**

`SMARTCARD_FLAG_TXE`  
`SMARTCARD_FLAG_TC`  
`SMARTCARD_FLAG_RXNE`  
`SMARTCARD_FLAG_IDLE`  
`SMARTCARD_FLAG_ORE`  
`SMARTCARD_FLAG_NE`  
`SMARTCARD_FLAG_FE`  
`SMARTCARD_FLAG_PE`

**SMARTCARD Interrupts Definition**

`SMARTCARD_IT_PE`  
`SMARTCARD_IT_TXE`  
`SMARTCARD_IT_TC`  
`SMARTCARD_IT_RXNE`  
`SMARTCARD_IT_IDLE`  
`SMARTCARD_IT_ERR`

**SMARTCARD Last Bit**

`SMARTCARD_LASTBIT_DISABLE`  
`SMARTCARD_LASTBIT_ENABLE`

**SMARTCARD Mode**

`SMARTCARD_MODE_RX`  
`SMARTCARD_MODE_TX`

SMARTCARD\_MODE\_TX\_RX

**SMARTCARD NACK State**

SMARTCARD\_NACK\_ENABLE

SMARTCARD\_NACK\_DISABLE

**SMARTCARD Parity**

SMARTCARD\_PARITY\_EVEN

SMARTCARD\_PARITY\_ODD

**SMARTCARD Prescaler**

SMARTCARD_PRESCALER_SYSCLK_DIV2	SYSCLK divided by 2
SMARTCARD_PRESCALER_SYSCLK_DIV4	SYSCLK divided by 4
SMARTCARD_PRESCALER_SYSCLK_DIV6	SYSCLK divided by 6
SMARTCARD_PRESCALER_SYSCLK_DIV8	SYSCLK divided by 8
SMARTCARD_PRESCALER_SYSCLK_DIV10	SYSCLK divided by 10
SMARTCARD_PRESCALER_SYSCLK_DIV12	SYSCLK divided by 12
SMARTCARD_PRESCALER_SYSCLK_DIV14	SYSCLK divided by 14
SMARTCARD_PRESCALER_SYSCLK_DIV16	SYSCLK divided by 16
SMARTCARD_PRESCALER_SYSCLK_DIV18	SYSCLK divided by 18
SMARTCARD_PRESCALER_SYSCLK_DIV20	SYSCLK divided by 20
SMARTCARD_PRESCALER_SYSCLK_DIV22	SYSCLK divided by 22
SMARTCARD_PRESCALER_SYSCLK_DIV24	SYSCLK divided by 24
SMARTCARD_PRESCALER_SYSCLK_DIV26	SYSCLK divided by 26
SMARTCARD_PRESCALER_SYSCLK_DIV28	SYSCLK divided by 28
SMARTCARD_PRESCALER_SYSCLK_DIV30	SYSCLK divided by 30
SMARTCARD_PRESCALER_SYSCLK_DIV32	SYSCLK divided by 32
SMARTCARD_PRESCALER_SYSCLK_DIV34	SYSCLK divided by 34
SMARTCARD_PRESCALER_SYSCLK_DIV36	SYSCLK divided by 36
SMARTCARD_PRESCALER_SYSCLK_DIV38	SYSCLK divided by 38
SMARTCARD_PRESCALER_SYSCLK_DIV40	SYSCLK divided by 40
SMARTCARD_PRESCALER_SYSCLK_DIV42	SYSCLK divided by 42
SMARTCARD_PRESCALER_SYSCLK_DIV44	SYSCLK divided by 44
SMARTCARD_PRESCALER_SYSCLK_DIV46	SYSCLK divided by 46
SMARTCARD_PRESCALER_SYSCLK_DIV48	SYSCLK divided by 48
SMARTCARD_PRESCALER_SYSCLK_DIV50	SYSCLK divided by 50
SMARTCARD_PRESCALER_SYSCLK_DIV52	SYSCLK divided by 52
SMARTCARD_PRESCALER_SYSCLK_DIV54	SYSCLK divided by 54
SMARTCARD_PRESCALER_SYSCLK_DIV56	SYSCLK divided by 56

SMARTCARD\_PRESCALER\_SYSCLK\_DIV58    SYSCLK divided by 58  
SMARTCARD\_PRESCALER\_SYSCLK\_DIV60    SYSCLK divided by 60  
SMARTCARD\_PRESCALER\_SYSCLK\_DIV62    SYSCLK divided by 62

***SMARTCARD Private Constants***

SMARTCARD\_TIMEOUT\_VALUE  
SMARTCARD\_IT\_MASK  
SMARTCARD\_DIV  
SMARTCARD\_DIVMANT  
SMARTCARD\_DIVFRAQ  
SMARTCARD\_BRR  
SMARTCARD\_CR1\_REG\_INDEX  
SMARTCARD\_CR3\_REG\_INDEX

***SMARTCARD Private Macros***

IS\_SMARTCARD\_WORD\_LENGTH  
IS\_SMARTCARD\_STOPBITS  
IS\_SMARTCARD\_PARITY  
IS\_SMARTCARD\_MODE  
IS\_SMARTCARD\_POLARITY  
IS\_SMARTCARD\_PHASE  
IS\_SMARTCARD\_LASTBIT  
IS\_SMARTCARD\_NACK\_STATE  
IS\_SMARTCARD\_BAUDRATE

***SMARTCARD Number of Stop Bits***

SMARTCARD\_STOPBITS\_0\_5  
SMARTCARD\_STOPBITS\_1\_5

***SMARTCARD Word Length***

SMARTCARD\_WORDLENGTH\_9B

## 40 HAL SPI Generic Driver

### 40.1 SPI Firmware driver registers structures

#### 40.1.1 SPI\_InitTypeDef

##### Data Fields

- *uint32\_t Mode*
- *uint32\_t Direction*
- *uint32\_t DataSize*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t NSS*
- *uint32\_t BaudRatePrescaler*
- *uint32\_t FirstBit*
- *uint32\_t TIMode*
- *uint32\_t CRCCalculation*
- *uint32\_t CRCPolynomial*

##### Field Documentation

- ***uint32\_t SPI\_InitTypeDef::Mode***  
Specifies the SPI operating mode. This parameter can be a value of [\*\*SPI\\_mode\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::Direction***  
Specifies the SPI Directional mode state. This parameter can be a value of [\*\*SPI\\_Direction\\_mode\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::DataSize***  
Specifies the SPI data size. This parameter can be a value of [\*\*SPI\\_data\\_size\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::CLKPolarity***  
Specifies the serial clock steady state. This parameter can be a value of [\*\*SPI\\_Clock\\_Polarity\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::CLKPhase***  
Specifies the clock active edge for the bit capture. This parameter can be a value of [\*\*SPI\\_Clock\\_Phase\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::NSS***  
Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of [\*\*SPI\\_Slave\\_Select\\_management\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::BaudRatePrescaler***  
Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of [\*\*SPI\\_BaudRate\\_Prescaler\*\*](#)  
**Note:** The communication clock is derived from the master clock. The slave clock does not need to be set
- ***uint32\_t SPI\_InitTypeDef::FirstBit***  
Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [\*\*SPI\\_MSB\\_LSB\\_transmission\*\*](#)
- ***uint32\_t SPI\_InitTypeDef::TIMode***  
Specifies if the TI mode is enabled or not. This parameter can be a value of [\*\*SPI\\_TI\\_mode\*\*](#)

- ***uint32\_t SPI\_InitTypeDef::CRCCalculation***  
Specifies if the CRC calculation is enabled or not. This parameter can be a value of [SPI\\_CRC\\_Calculation](#)
- ***uint32\_t SPI\_InitTypeDef::CRCPolynomial***  
Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min\_Data = 0 and Max\_Data = 65535

## 40.1.2 \_\_SPI\_HandleTypeDef

### Data Fields

- ***SPI\_TypeDef \* Instance***
- ***SPI\_InitTypeDef Init***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferSize***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***void(\* RxISR***
- ***void(\* TxISR***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_SPI\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

### Field Documentation

- ***SPI\_TypeDef\* \_\_SPI\_HandleTypeDef::Instance***
- ***SPI\_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 uint32\_t \_\_SPI\_HandleTypeDef::ErrorCode***

## 40.2 SPI Firmware driver API description

### 40.2.1 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.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
  - a. Master 2Lines RxOnly
  - b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL\_SPI\_DMAPause()/ HAL\_SPI\_DMAStop() only under the SPI callbacks

### 40.2.2 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.

This section contains the following APIs:

- [\*HAL\\_SPI\\_Init\(\)\*](#)
- [\*HAL\\_SPI\\_DeInit\(\)\*](#)
- [\*HAL\\_SPI\\_MspInit\(\)\*](#)
- [\*HAL\\_SPI\\_MspDeInit\(\)\*](#)

### 40.2.3 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\_TxRx\_CpltCallback() 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. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- [\*HAL\\_SPI\\_Transmit\(\)\*](#)
- [\*HAL\\_SPI\\_Receive\(\)\*](#)
- [\*HAL\\_SPI\\_TransmitReceive\(\)\*](#)
- [\*HAL\\_SPI\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_SPI\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_SPI\\_TransmitReceive\\_IT\(\)\*](#)
- [\*HAL\\_SPI\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_SPI\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_SPI\\_TransmitReceive\\_DMA\(\)\*](#)
- [\*HAL\\_SPI\\_DMAPause\(\)\*](#)
- [\*HAL\\_SPI\\_DMAResume\(\)\*](#)
- [\*HAL\\_SPI\\_DMAStop\(\)\*](#)
- [\*HAL\\_SPI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SPI\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_TxRx\\_CpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_TxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_TxRxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_SPI\\_ErrorCallback\(\)\*](#)

### 40.2.4 Peripheral State and Errors functions

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

- `HAL_SPI_GetState()` API can be helpful to check in run-time the state of the SPI peripheral
- `HAL_SPI_GetError()` check in run-time Errors occurring during communication

This section contains the following APIs:

- `HAL\_SPI\_GetState\(\)`
- `HAL\_SPI\_GetError\(\)`

#### 40.2.5 `HAL_SPI_Init`

Function Name	<code>HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)</code>
Function Description	Initializes the SPI according to the specified parameters in the <code>SPI_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <code>hspi</code>: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 40.2.6 `HAL_SPI_DeInit`

Function Name	<code>HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)</code>
Function Description	DeInitializes the SPI peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <code>hspi</code>: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 40.2.7 `HAL_SPI_MspInit`

Function Name	<code>void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)</code>
Function Description	SPI MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <code>hspi</code>: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 40.2.8 `HAL_SPI_MspDeInit`

Function Name	<code>void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)</code>
Function Description	SPI MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <code>hspi</code>: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 40.2.9 `HAL_SPI_Transmit`

Function Name	<code>HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
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>HAL status</li> </ul>

#### 40.2.10 HAL\_SPI\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * 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>HAL status</li> </ul>

#### 40.2.11 HAL\_SPI\_TransmitReceive

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive (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>HAL status</li> </ul>

#### 40.2.12 HAL\_SPI\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_IT (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>HAL status</li> </ul>

#### 40.2.13 HAL\_SPI\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_IT (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>HAL status</li> </ul>

#### 40.2.14 HAL\_SPI\_TransmitReceive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT(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>HAL status</li> </ul>

#### 40.2.15 HAL\_SPI\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit_DMA(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>HAL status</li> </ul>

#### 40.2.16 HAL\_SPI\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Receive_DMA(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> <li><b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

Notes

- When the CRC feature is enabled the pData Length must be Size + 1.

#### 40.2.17 HAL\_SPI\_TransmitReceive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (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> <li><b>pRxData:</b> pointer to reception data buffer</li> <li><b>Size:</b> amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the CRC feature is enabled the pRxData Length must be Size + 1</li> </ul>

#### 40.2.18 HAL\_SPI\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)</b>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 40.2.19 HAL\_SPI\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 40.2.20 HAL\_SPI\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 40.2.21 HAL\_SPI\_IRQHandler

Function Name	<b>void HAL_SPI_IRQHandler (SPI_HandleTypeDef * 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</li> </ul>

the configuration information for SPI module.

Return values • HAL status

#### 40.2.22 HAL\_SPI\_TxCpltCallback

Function Name **void HAL\_SPI\_TxCpltCallback (SPI\_HandleTypeDef \* hspi)**

Function Description Tx Transfer completed callbacks.

Parameters • **hspi:** pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.

Return values • None

#### 40.2.23 HAL\_SPI\_RxCpltCallback

Function Name **void HAL\_SPI\_RxCpltCallback (SPI\_HandleTypeDef \* hspi)**

Function Description Rx Transfer completed callbacks.

Parameters • **hspi:** pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.

Return values • None

#### 40.2.24 HAL\_SPI\_TxRxCpltCallback

Function Name **void HAL\_SPI\_TxRxCpltCallback (SPI\_HandleTypeDef \* hspi)**

Function Description Tx and Rx Transfer completed callbacks.

Parameters • **hspi:** pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.

Return values • None

#### 40.2.25 HAL\_SPI\_TxHalfCpltCallback

Function Name **void HAL\_SPI\_TxHalfCpltCallback (SPI\_HandleTypeDef \* hspi)**

Function Description Tx Half Transfer completed callbacks.

Parameters • **hspi:** pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.

Return values • None

#### 40.2.26 HAL\_SPI\_RxHalfCpltCallback

Function Name **void HAL\_SPI\_RxHalfCpltCallback (SPI\_HandleTypeDef \* hspi)**

Function Description Rx Half Transfer completed callbacks.

Parameters • **hspi:** pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.

Return values • None

#### 40.2.27 HAL\_SPI\_TxRxHalfCpltCallback

Function Name	<b>void HAL_SPI_TxRxHalfCpltCallback (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>

#### 40.2.28 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>

#### 40.2.29 HAL\_SPI\_GetState

Function Name	<b>HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)</b>
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>• HAL state</li> </ul>

#### 40.2.30 HAL\_SPI\_GetError

Function Name	<b>uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)</b>
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>• SPI Error Code</li> </ul>

### 40.3 SPI Firmware driver defines

#### 40.3.1 SPI

**SPI BaudRate Prescaler**

SPI\_BAUDRATEPRESCALER\_2  
 SPI\_BAUDRATEPRESCALER\_4  
 SPI\_BAUDRATEPRESCALER\_8  
 SPI\_BAUDRATEPRESCALER\_16  
 SPI\_BAUDRATEPRESCALER\_32  
 SPI\_BAUDRATEPRESCALER\_64  
 SPI\_BAUDRATEPRESCALER\_128

SPI\_BAUDRATEPRESCALER\_256

**SPI Clock Phase**

SPI\_PHASE\_1EDGE

SPI\_PHASE\_2EDGE

**SPI Clock Polarity**

SPI\_POLARITY\_LOW

SPI\_POLARITY\_HIGH

**SPI CRC Calculation**

SPI\_CRCCALCULATION\_DISABLE

SPI\_CRCCALCULATION\_ENABLE

**SPI Data Size**

SPI\_DATASIZE\_8BIT

SPI\_DATASIZE\_16BIT

**SPI Direction Mode**

SPI\_DIRECTION\_2LINES

SPI\_DIRECTION\_2LINES\_RXONLY

SPI\_DIRECTION\_1LINE

**SPI Error Code**

HAL\_SPI\_ERROR\_NONE No error

HAL\_SPI\_ERROR\_MODF MODF error

HAL\_SPI\_ERROR\_CRC CRC error

HAL\_SPI\_ERROR\_OVR OVR error

HAL\_SPI\_ERROR\_FRE FRE error

HAL\_SPI\_ERROR\_DMA DMA transfer error

HAL\_SPI\_ERROR\_FLAG Flag: RXNE,TXE, BSY

**SPI Exported Macros**

\_\_HAL\_SPI\_RESET\_HANDLE\_STATE **Description:**

- Reset SPI handle state.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

\_\_HAL\_SPI\_ENABLE\_IT

**Description:**

- Enable or disable the specified SPI interrupts.

**Parameters:**

- HANDLE: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- INTERRUPT: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPI\_IT\_TXE: Tx buffer empty interrupt enable
  - SPI\_IT\_RXNE: RX buffer not empty interrupt enable
  - SPI\_IT\_ERR: Error interrupt enable

**Return value:**

- None

\_HAL\_SPI\_DISABLE\_IT  
\_HAL\_SPI\_GET\_IT\_SOURCE

**Description:**

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

**Parameters:**

- HANDLE: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- INTERRUPT: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI\_IT\_TXE: Tx buffer empty interrupt enable
  - SPI\_IT\_RXNE: RX buffer not empty interrupt enable
  - SPI\_IT\_ERR: Error interrupt enable

**Return value:**

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

\_HAL\_SPI\_GET\_FLAG

**Description:**

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

**Parameters:**

- HANDLE: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- FLAG: specifies the flag to check. This parameter can be one of the following values:
  - SPI\_FLAG\_RXNE: Receive buffer not empty flag
  - SPI\_FLAG\_TXE: Transmit buffer empty flag
  - SPI\_FLAG\_CRCERR: CRC error flag

- SPI\_FLAG\_MODF: Mode fault flag
- SPI\_FLAG\_OVR: Overrun flag
- SPI\_FLAG\_BSY: Busy flag
- SPI\_FLAG\_FRE: Frame format error flag

**Return value:**

- The: new state of \_\_FLAG\_\_ (TRUE or FALSE).

[\\_\\_HAL\\_SPI\\_CLEAR\\_CRCERRFLAG](#)

**Description:**

- Clear the SPI CRCERR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

[\\_\\_HAL\\_SPI\\_CLEAR\\_MODFFLAG](#)

**Description:**

- Clear the SPI MODF pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

[\\_\\_HAL\\_SPI\\_CLEAR\\_OVRFAG](#)

**Description:**

- Clear the SPI OVR pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

[\\_\\_HAL\\_SPI\\_CLEAR\\_FREFLAG](#)

**Description:**

- Clear the SPI FRE pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SPI handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

<code>__HAL_SPI_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Enable SPI.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: specifies the SPI Handle.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None</li></ul>
<code>__HAL_SPI_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>• Disable SPI.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li>• <code>__HANDLE__</code>: specifies the SPI Handle.</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>• None</li></ul>

***SPI Flags Definition***

`SPI_FLAG_RXNE`  
`SPI_FLAG_TXE`  
`SPI_FLAG_CRCERR`  
`SPI_FLAG_MODF`  
`SPI_FLAG_OVR`  
`SPI_FLAG_BSY`  
`SPI_FLAG_FRE`

***SPI Interrupt Definition***

`SPI_IT_TXE`  
`SPI_IT_RXNE`  
`SPI_IT_ERR`

***SPI Mode***

`SPI_MODE_SLAVE`  
`SPI_MODE_MASTER`

***SPI MSB LSB Transssmission***

`SPI_FIRSTBIT_MSB`  
`SPI_FIRSTBIT_LSB`

***SPI Private Macros***

`IS_SPI_MODE`  
`IS_SPI_DIRECTION_MODE`  
`IS_SPI_DIRECTION_2LINES_OR_1LINE`  
`IS_SPI_DIRECTION_2LINES`  
`IS_SPI_DATASIZE`

IS\_SPI\_CPOL  
IS\_SPI\_CPHA  
IS\_SPI\_NSS  
IS\_SPI\_BAUDRATE\_PRESCALER  
IS\_SPI\_FIRST\_BIT  
IS\_SPI\_TIMODE  
IS\_SPI\_CRC\_CALCULATION  
IS\_SPI\_CRC\_POLYNOMIAL  
SPI\_1LINE\_TX  
SPI\_1LINE\_RX  
SPI\_RESET\_CRC  
**SPI Slave Select Management**  
SPI\_NSS\_SOFT  
SPI\_NSS\_HARD\_INPUT  
SPI\_NSS\_HARD\_OUTPUT  
**SPI TI Mode**  
SPI\_TIMODE\_DISABLE  
SPI\_TIMODE\_ENABLE

## 41 HAL SRAM Generic Driver

### 41.1 SRAM Firmware driver registers structures

#### 41.1.1 SRAM\_HandleTypeDef

##### Data Fields

- *FSMC\_NORSRAM\_TypeDef \* Instance*
- *FSMC\_NORSRAM\_EXTENDED\_TypeDef \* Extended*
- *FSMC\_NORSRAM\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_SRAM\_StateTypeDef State*
- *DMA\_HandleTypeDef \* hdma*

##### Field Documentation

- ***FSMC\_NORSRAM\_TypeDef\* SRAM\_HandleTypeDef::Instance***  
Register base address
- ***FSMC\_NORSRAM\_EXTENDED\_TypeDef\* SRAM\_HandleTypeDef::Extended***  
Extended mode register base address
- ***FSMC\_NORSRAM\_InitTypeDef SRAM\_HandleTypeDef::Init***  
SRAM device control configuration parameters
- ***HAL\_LockTypeDef SRAM\_HandleTypeDef::Lock***  
SRAM locking object
- ***\_\_IO HAL\_SRAM\_StateTypeDef SRAM\_HandleTypeDef::State***  
SRAM device access state
- ***DMA\_HandleTypeDef\* SRAM\_HandleTypeDef::hdma***  
Pointer DMA handler

### 41.2 SRAM Firmware driver API description

#### 41.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FSMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FSMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM\_HandleTypeDef handle structure, for example:  
`SRAM_HandleTypeDef hsramp;` and:
  - Fill the SRAM\_HandleTypeDef handle "Init" field with the allowed values of the structure member.
  - Fill the SRAM\_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
  - Fill the SRAM\_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FSMC\_NORSRAM\_TimingTypeDef structures, for both normal and extended mode timings; for example: `FSMC_NORSRAM_TimingTypeDef Timing` and

- FSMC\_NORSRAM\_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL\_SRAM\_Init(). This function performs the following sequence:
    - a. MSP hardware layer configuration using the function HAL\_SRAM\_MspInit()
    - b. Control register configuration using the FSMC NORSRAM interface function FSMC\_NORSRAM\_Init()
    - c. Timing register configuration using the FSMC NORSRAM interface function FSMC\_NORSRAM\_Timing\_Init()
    - d. Extended mode Timing register configuration using the FSMC NORSRAM interface function FSMC\_NORSRAM\_Extended\_Timing\_Init()
    - e. Enable the SRAM device using the macro \_\_FSMC\_NORSRAM\_ENABLE()
  4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
    - HAL\_SRAM\_Read()/HAL\_SRAM\_Write() for polling read/write access
    - HAL\_SRAM\_Read\_DMA()/HAL\_SRAM\_Write\_DMA() for DMA read/write transfer
  5. You can also control the SRAM device by calling the control APIs HAL\_SRAM\_WriteOperation\_Enable()/ HAL\_SRAM\_WriteOperation\_Disable() to respectively enable/disable the SRAM write operation
  6. You can continuously monitor the SRAM device HAL state by calling the function HAL\_SRAM\_GetState()

#### 41.2.2

#### SRAM Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

This section contains the following APIs:

- [\*HAL\\_SRAM\\_Init\(\)\*](#)
- [\*HAL\\_SRAM\\_DeInit\(\)\*](#)
- [\*HAL\\_SRAM\\_MspInit\(\)\*](#)
- [\*HAL\\_SRAM\\_MspDeInit\(\)\*](#)
- [\*HAL\\_SRAM\\_DMA\\_XferCpltCallback\(\)\*](#)
- [\*HAL\\_SRAM\\_DMA\\_XferErrorCallback\(\)\*](#)

#### 41.2.3

#### SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

This section contains the following APIs:

- [\*HAL\\_SRAM\\_Read\\_8b\(\)\*](#)
- [\*HAL\\_SRAM\\_Write\\_8b\(\)\*](#)
- [\*HAL\\_SRAM\\_Read\\_16b\(\)\*](#)
- [\*HAL\\_SRAM\\_Write\\_16b\(\)\*](#)
- [\*HAL\\_SRAM\\_Read\\_32b\(\)\*](#)
- [\*HAL\\_SRAM\\_Write\\_32b\(\)\*](#)
- [\*HAL\\_SRAM\\_Read\\_DMA\(\)\*](#)
- [\*HAL\\_SRAM\\_Write\\_DMA\(\)\*](#)

#### 41.2.4

#### SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

This section contains the following APIs:

- `HAL_SRAM_WriteOperation_Enable()`
- `HAL_SRAM_WriteOperation_Disable()`

#### 41.2.5 SRAM State functions

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

This section contains the following APIs:

- `HAL_SRAM_GetState()`

#### 41.2.6 HAL\_SRAM\_Init

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_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>• HAL status</li> </ul>

#### 41.2.7 HAL\_SRAM\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_DelInit (SRAM_HandleTypeDef * 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>• HAL status</li> </ul>

#### 41.2.8 HAL\_SRAM\_MspInit

Function Name	<code>void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)</code>
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>

#### 41.2.9 HAL\_SRAM\_MspDelInit

Function Name	<code>void HAL_SRAM_MspDelInit (SRAM_HandleTypeDef * hsram)</code>
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>

#### 41.2.10 HAL\_SRAM\_DMA\_XferCpltCallback

Function Name	<code>void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)</code>
Function Description	DMA transfer complete callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma:</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>

#### 41.2.11 HAL\_SRAM\_DMA\_XferErrorCallback

Function Name	<code>void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)</code>
Function Description	DMA transfer complete error callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma:</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>

#### 41.2.12 HAL\_SRAM\_Read\_8b

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)</code>
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>• HAL status</li> </ul>

#### 41.2.13 HAL\_SRAM\_Write\_8b

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)</code>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 41.2.14 HAL\_SRAM\_Read\_16b

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t</code>
---------------	---

**\* pDstBuffer, uint32\_t BufferSize)**

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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 41.2.15 HAL\_SRAM\_Write\_16b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Write_16b (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> <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>• HAL status</li> </ul>

#### 41.2.16 HAL\_SRAM\_Read\_32b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)</b>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 41.2.17 HAL\_SRAM\_Write\_32b

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)</b>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 41.2.18 HAL\_SRAM\_Read\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 41.2.19 HAL\_SRAM\_Write\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 41.2.20 HAL\_SRAM\_WriteOperation\_Enable

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)</code>
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>• HAL status</li> </ul>

#### 41.2.21 HAL\_SRAM\_WriteOperation\_Disable

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)</code>
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>• HAL status</li> </ul>

#### 41.2.22 HAL\_SRAM\_GetState

Function Name	<code>HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)</code>
---------------	---

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>• HAL state</li></ul>

## 41.3 SRAM Firmware driver defines

### 41.3.1 SRAM

#### *SRAM Exported Macros*

`_HAL_SRAM_RESET_HANDLE_STATE` **Description:**

- Reset SRAM handle state.

**Parameters:**

- `_HANDLE_`: SRAM handle

**Return value:**

- None

## 42 HAL TIM Generic Driver

### 42.1 TIM Firmware driver registers structures

#### 42.1.1 TIM\_Base\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t CounterMode*
- *uint32\_t Period*
- *uint32\_t ClockDivision*
- *uint32\_t RepetitionCounter*

##### Field Documentation

- ***uint32\_t TIM\_Base\_InitTypeDef::Prescaler***  
Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t TIM\_Base\_InitTypeDef::CounterMode***  
Specifies the counter mode. This parameter can be a value of [\*\*TIM\\_Counter\\_Mode\*\*](#)
- ***uint32\_t TIM\_Base\_InitTypeDef::Period***  
Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- ***uint32\_t TIM\_Base\_InitTypeDef::ClockDivision***  
Specifies the clock division. This parameter can be a value of [\*\*TIM\\_ClockDivision\*\*](#)
- ***uint32\_t TIM\_Base\_InitTypeDef::RepetitionCounter***  
Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to the number of PWM periods in edge-aligned mode the number of half PWM period in center-aligned mode This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.  
**Note:**This parameter is valid only for TIM1 and TIM8.

#### 42.1.2 TIM\_OC\_InitTypeDef

##### Data Fields

- *uint32\_t OCMode*
- *uint32\_t Pulse*
- *uint32\_t OCIdleState*
- *uint32\_t OCNPolarity*
- *uint32\_t OCNPolarity*
- *uint32\_t OCFastMode*
- *uint32\_t OCIdleState*
- *uint32\_t OCIdleState*

### Field Documentation

- ***uint32\_t TIM\_OC\_InitTypeDef::OCMode***  
Specifies the TIM mode. This parameter can be a value of  
***TIM\_Output\_Compare\_and\_PWM\_modes***
- ***uint32\_t TIM\_OC\_InitTypeDef::Pulse***  
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t TIM\_OC\_InitTypeDef::OCPolarity***  
Specifies the output polarity. This parameter can be a value of  
***TIM\_Output\_Compare\_Polarity***
- ***uint32\_t TIM\_OC\_InitTypeDef::OCNPolarity***  
Specifies the complementary output polarity. This parameter can be a value of  
***TIM\_Output\_Compare\_N\_Polarity***  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OC\_InitTypeDef::OCFastMode***  
Specifies the Fast mode state. This parameter can be a value of  
***TIM\_Output\_Fast\_State***  
**Note:**This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32\_t TIM\_OC\_InitTypeDef::OCIdleState***  
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of  
***TIM\_Output\_Compare\_Idle\_State***  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OC\_InitTypeDef::OCNIdleState***  
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of  
***TIM\_Output\_Compare\_N\_Idle\_State***  
**Note:**This parameter is valid only for TIM1 and TIM8.

### 42.1.3 TIM\_OnePulse\_InitTypeDef

#### Data Fields

- ***uint32\_t OCMode***
- ***uint32\_t Pulse***
- ***uint32\_t OCPolarity***
- ***uint32\_t OCNPolarity***
- ***uint32\_t OCIdleState***
- ***uint32\_t OCNIdleState***
- ***uint32\_t IC\_Polarity***
- ***uint32\_t IC\_Selection***
- ***uint32\_t IC\_Filter***

#### Field Documentation

- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCMode***  
Specifies the TIM mode. This parameter can be a value of  
***TIM\_Output\_Compare\_and\_PWM\_modes***
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::Pulse***  
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF

- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OC\_Polarity***  
Specifies the output polarity. This parameter can be a value of ***TIM\_Output\_Compare\_Polarity***
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OC\_N\_Polarity***  
Specifies the complementary output polarity. This parameter can be a value of ***TIM\_Output\_Compare\_N\_Polarity***  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OC\_Idle\_State***  
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM\_Output\_Compare\_Idle\_State***  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCN\_Idle\_State***  
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM\_Output\_Compare\_N\_Idle\_State***  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::IC\_Polarity***  
Specifies the active edge of the input signal. This parameter can be a value of ***TIM\_Input\_Capture\_Polarity***
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::IC\_Selection***  
Specifies the input. This parameter can be a value of ***TIM\_Input\_Capture\_Selection***
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::IC\_Filter***  
Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

#### 42.1.4 TIM\_IC\_InitTypeDef

##### 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

#### 42.1.5 TIM\_Encoder\_InitTypeDef

**Data Fields**

- *uint32\_t EncoderMode*
- *uint32\_t IC1Polarity*
- *uint32\_t IC1Selection*
- *uint32\_t IC1Prescaler*
- *uint32\_t IC1Filter*
- *uint32\_t IC2Polarity*
- *uint32\_t IC2Selection*
- *uint32\_t IC2Prescaler*
- *uint32\_t IC2Filter*

**Field Documentation**

- *uint32\_t TIM\_Encoder\_InitTypeDef::EncoderMode*  
Specifies the active edge of the input signal. This parameter can be a value of [\*\*TIM\\_Encoder\\_Mode\*\*](#)
- *uint32\_t TIM\_Encoder\_InitTypeDef::IC1Polarity*  
Specifies the active edge of the input signal. This parameter can be a value of [\*\*TIM\\_Input\\_Capture\\_Polarity\*\*](#)
- *uint32\_t TIM\_Encoder\_InitTypeDef::IC1Selection*  
Specifies the input. This parameter can be a value of [\*\*TIM\\_Input\\_Capture\\_Selection\*\*](#)
- *uint32\_t TIM\_Encoder\_InitTypeDef::IC1Prescaler*  
Specifies the Input Capture Prescaler. This parameter can be a value of [\*\*TIM\\_Input\\_Capture\\_Prescaler\*\*](#)
- *uint32\_t TIM\_Encoder\_InitTypeDef::IC1Filter*  
Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF
- *uint32\_t TIM\_Encoder\_InitTypeDef::IC2Polarity*  
Specifies the active edge of the input signal. This parameter can be a value of [\*\*TIM\\_Input\\_Capture\\_Polarity\*\*](#)
- *uint32\_t TIM\_Encoder\_InitTypeDef::IC2Selection*  
Specifies the input. This parameter can be a value of [\*\*TIM\\_Input\\_Capture\\_Selection\*\*](#)
- *uint32\_t TIM\_Encoder\_InitTypeDef::IC2Prescaler*  
Specifies the Input Capture Prescaler. This parameter can be a value of [\*\*TIM\\_Input\\_Capture\\_Prescaler\*\*](#)
- *uint32\_t TIM\_Encoder\_InitTypeDef::IC2Filter*  
Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

**42.1.6 TIM\_ClockConfigTypeDef****Data Fields**

- *uint32\_t ClockSource*
- *uint32\_t ClockPolarity*
- *uint32\_t ClockPrescaler*
- *uint32\_t ClockFilter*

**Field Documentation**

- ***uint32\_t TIM\_ClockConfigTypeDef::ClockSource***  
TIM clock sources. This parameter can be a value of [\*\*TIM\\_Clock\\_Source\*\*](#)
- ***uint32\_t TIM\_ClockConfigTypeDef::ClockPolarity***  
TIM clock polarity. This parameter can be a value of [\*\*TIM\\_Clock\\_Polarity\*\*](#)
- ***uint32\_t TIM\_ClockConfigTypeDef::ClockPrescaler***  
TIM clock prescaler. This parameter can be a value of [\*\*TIM\\_Clock\\_Prescaler\*\*](#)
- ***uint32\_t TIM\_ClockConfigTypeDef::ClockFilter***  
TIM clock filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

#### 42.1.7 **TIM\_ClearInputConfigTypeDef**

##### Data Fields

- ***uint32\_t ClearInputState***
- ***uint32\_t ClearInputSource***
- ***uint32\_t ClearInputPolarity***
- ***uint32\_t ClearInputPrescaler***
- ***uint32\_t ClearInputFilter***

##### Field Documentation

- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputState***  
TIM clear Input state. This parameter can be ENABLE or DISABLE
- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputSource***  
TIM clear Input sources. This parameter can be a value of [\*\*TIM\\_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

#### 42.1.8 **TIM\_SlaveConfigTypeDef**

##### Data Fields

- ***uint32\_t SlaveMode***
- ***uint32\_t InputTrigger***
- ***uint32\_t TriggerPolarity***
- ***uint32\_t TriggerPrescaler***
- ***uint32\_t TriggerFilter***

##### Field Documentation

- ***uint32\_t TIM\_SlaveConfigTypeDef::SlaveMode***  
Slave mode selection This parameter can be a value of [\*\*TIM\\_Slave\\_Mode\*\*](#)

- ***uint32\_t TIM\_SlaveConfigTypeDef::InputTrigger***  
Input Trigger source This parameter can be a value of [\*\*TIM\\_Trigger\\_Selection\*\*](#)
- ***uint32\_t TIM\_SlaveConfigTypeDef::TriggerPolarity***  
Input Trigger polarity This parameter can be a value of [\*\*TIM\\_Trigger\\_Polarity\*\*](#)
- ***uint32\_t TIM\_SlaveConfigTypeDef::TriggerPrescaler***  
Input trigger prescaler This parameter can be a value of [\*\*TIM\\_Trigger\\_Prescaler\*\*](#)
- ***uint32\_t TIM\_SlaveConfigTypeDef::TriggerFilter***  
Input trigger filter This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

## 42.1.9 TIM\_HandleTypeDef

### Data Fields

- ***TIM\_TypeDef \* Instance***
- ***TIM\_Base\_InitTypeDef Init***
- ***HAL\_TIM\_ActiveChannel Channel***
- ***DMA\_HandleTypeDef \* hdma***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_TIM\_StateTypeDef State***

### Field Documentation

- ***TIM\_TypeDef\* TIM\_HandleTypeDef::Instance***  
Register base address
- ***TIM\_Base\_InitTypeDef TIM\_HandleTypeDef::Init***  
TIM Time Base required parameters
- ***HAL\_TIM\_ActiveChannel TIM\_HandleTypeDef::Channel***  
Active channel
- ***DMA\_HandleTypeDef\* TIM\_HandleTypeDef::hdma[7]***  
DMA Handlers array This array is accessed by a [\*\*DMA\\_Handle\\_index\*\*](#)
- ***HAL\_LockTypeDef TIM\_HandleTypeDef::Lock***  
Locking object
- ***\_\_IO HAL\_TIM\_StateTypeDef TIM\_HandleTypeDef::State***  
TIM operation state

## 42.2 TIM Firmware driver API description

### 42.2.1 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

## 42.2.2 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()

## 42.2.3 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.

This section contains the following APIs:

- [\*HAL\\_TIM\\_Base\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_Base\\_Stop\\_DMA\(\)\*](#)

#### 42.2.4 Time Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the Time Output Compare.
- Stop the Time Output Compare.
- Start the Time Output Compare and enable interrupt.
- Stop the Time Output Compare and disable interrupt.
- Start the Time Output Compare and enable DMA transfer.
- Stop the Time Output Compare and disable DMA transfer.

This section contains the following APIs:

- [\*HAL\\_TIM\\_OC\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_Stop\\_DMA\(\)\*](#)

#### 42.2.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM OPWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.

- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.

This section contains the following APIs:

- [\*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\(\)\*](#)

#### 42.2.6 Time Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the Time Input Capture.
- Stop the Time Input Capture.
- Start the Time Input Capture and enable interrupt.
- Stop the Time Input Capture and disable interrupt.
- Start the Time Input Capture and enable DMA transfer.
- Stop the Time Input Capture and disable DMA transfer.

This section contains the following APIs:

- [\*HAL\\_TIM\\_IC\\_Init\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_Stop\\_DMA\(\)\*](#)

#### 42.2.7 Time One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the Time One Pulse.
- Stop the Time One Pulse.
- Start the Time One Pulse and enable interrupt.
- Stop the Time One Pulse and disable interrupt.
- Start the Time One Pulse and enable DMA transfer.
- Stop the Time One Pulse and disable DMA transfer.

This section contains the following APIs:

- [\*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\(\)\*](#)

#### 42.2.8 Time Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the Time Encoder.
- Stop the Time Encoder.
- Start the Time Encoder and enable interrupt.
- Stop the Time Encoder and disable interrupt.
- Start the Time Encoder and enable DMA transfer.
- Stop the Time Encoder and disable DMA transfer.

This section contains the following APIs:

- [\*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\(\)\*](#)

#### 42.2.9 IRQ handler management

This section provides Timer IRQ handler function.

This section contains the following APIs:

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

#### 42.2.10 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.

This section contains the following APIs:

- [\*HAL\\_TIM\\_OC\\_ConfigChannel\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_ConfigChannel\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_ConfigChannel\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_ConfigChannel\(\)\*](#)

- [\*HAL\\_TIM\\_DMABurst\\_WriteStart\(\)\*](#)
- [\*HAL\\_TIM\\_DMABurst\\_WriteStop\(\)\*](#)
- [\*HAL\\_TIM\\_DMABurst\\_ReadStart\(\)\*](#)
- [\*HAL\\_TIM\\_DMABurst\\_ReadStop\(\)\*](#)
- [\*HAL\\_TIM\\_GenerateEvent\(\)\*](#)
- [\*HAL\\_TIM\\_ConfigOCrefClear\(\)\*](#)
- [\*HAL\\_TIM\\_ConfigClockSource\(\)\*](#)
- [\*HAL\\_TIM\\_ConfigT1Input\(\)\*](#)
- [\*HAL\\_TIM\\_SlaveConfigSynchronization\(\)\*](#)
- [\*HAL\\_TIM\\_SlaveConfigSynchronization\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_ReadCapturedValue\(\)\*](#)

#### 42.2.11 TIM Callbacks functions

This section provides TIM callback functions:

- Timer Period elapsed callback
- Timer Output Compare callback
- Timer Input capture callback
- Timer Trigger callback
- Timer Error callback

This section contains the following APIs:

- [\*HAL\\_TIM\\_PeriodElapsedCallback\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_DelayElapsedCallback\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_CaptureCallback\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_PulseFinishedCallback\(\)\*](#)
- [\*HAL\\_TIM\\_TriggerCallback\(\)\*](#)
- [\*HAL\\_TIM\\_ErrorCallback\(\)\*](#)

#### 42.2.12 Peripheral State functions

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

This section contains the following APIs:

- [\*HAL\\_TIM\\_Base\\_GetState\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_GetState\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_GetState\(\)\*](#)
- [\*HAL\\_TIM\\_IC\\_GetState\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_GetState\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_GetState\(\)\*](#)

#### 42.2.13 HAL\_TIM\_Base\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Time base Unit 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>• HAL status</li> </ul>

#### 42.2.14 HAL\_TIM\_Base\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)</b>
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>• HAL status</li> </ul>

#### 42.2.15 HAL\_TIM\_Base\_MspInit

Function Name	<b>void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)</b>
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>

#### 42.2.16 HAL\_TIM\_Base\_MspDeInit

Function Name	<b>void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)</b>
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>

#### 42.2.17 HAL\_TIM\_Base\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start (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>• HAL status</li> </ul>

#### 42.2.18 HAL\_TIM\_Base\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop (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>• HAL status</li> </ul>

#### 42.2.19 HAL\_TIM\_Base\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start_IT</b>
---------------	--

**(TIM\_HandleTypeDef \* htim)**

Function Description Starts the TIM Base generation in interrupt mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- HAL status

#### 42.2.20 HAL\_TIM\_Base\_Stop\_IT

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Base\_Stop\_IT  
(TIM\_HandleTypeDef \* htim)**

Function Description Stops the TIM Base generation in interrupt mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- HAL status

#### 42.2.21 HAL\_TIM\_Base\_Start\_DMA

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Base\_Start\_DMA  
(TIM\_HandleTypeDef \* htim, uint32\_t \* pData, uint16\_t Length)**

Function Description Starts the TIM Base generation in DMA mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to peripheral.

Return values

- HAL status

#### 42.2.22 HAL\_TIM\_Base\_Stop\_DMA

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Base\_Stop\_DMA  
(TIM\_HandleTypeDef \* htim)**

Function Description Stops the TIM Base generation in DMA mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- HAL status

#### 42.2.23 HAL\_TIM\_OC\_Init

Function Name **HAL\_StatusTypeDef HAL\_TIM\_OC\_Init (TIM\_HandleTypeDef \* htim)**

Function Description Initializes the TIM Output Compare according to the specified parameters in the TIM\_HandleTypeDef and create the associated handle.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- HAL status

#### 42.2.24 HAL\_TIM\_OC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM 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>• HAL status</li> </ul>

#### 42.2.25 HAL\_TIM\_OC\_MspInit

Function Name	<b>void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Output Compare 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>

#### 42.2.26 HAL\_TIM\_OC\_MspDeInit

Function Name	<b>void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Output Compare 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>

#### 42.2.27 HAL\_TIM\_OC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Starts the TIM Output Compare signal 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> <li>• <b>Channel:</b> TIM Channel to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.28 HAL\_TIM\_OC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM Output Compare signal 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> <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 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

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

#### 42.2.29 HAL\_TIM\_OC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.30 HAL\_TIM\_OC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.31 HAL\_TIM\_OC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> <li>• <b>pData:</b> The source Buffer address.</li> </ul>

- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values • HAL status

#### 42.2.32 HAL\_TIM\_OC\_Stop\_DMA

Function Name **HAL\_StatusTypeDef HAL\_TIM\_OC\_Stop\_DMA (TIM\_HandleTypeDef \* htim, uint32\_t Channel)**

Function Description Stops the TIM Output Compare signal generation in DMA mode.

Parameters
 

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channel to be disabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values • HAL status

#### 42.2.33 HAL\_TIM\_PWM\_Init

Function Name **HAL\_StatusTypeDef HAL\_TIM\_PWM\_Init (TIM\_HandleTypeDef \* htim)**

Function Description Initializes the TIM PWM Time Base according to the specified parameters in the TIM\_HandleTypeDef and create the associated handle.

Parameters
 

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values • HAL status

#### 42.2.34 HAL\_TIM\_PWM\_DeInit

Function Name **HAL\_StatusTypeDef HAL\_TIM\_PWM\_DeInit (TIM\_HandleTypeDef \* htim)**

Function Description DeInitializes the TIM peripheral.

Parameters
 

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values • HAL status

#### 42.2.35 HAL\_TIM\_PWM\_MspInit

Function Name **void HAL\_TIM\_PWM\_MspInit (TIM\_HandleTypeDef \* htim)**

Function Description Initializes the TIM PWM MSP.

Parameters
 

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values • None

#### 42.2.36 HAL\_TIM\_PWM\_MspDeInit



Function Name	<b>void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	Deinitializes TIM PWM 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>

#### 42.2.37 HAL\_TIM\_PWM\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * 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 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.38 HAL\_TIM\_PWM\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * 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 TIM_HandleTypeDef 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.39 HAL\_TIM\_PWM\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * 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 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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.40 HAL\_TIM\_PWM\_Stop\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the PWM 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 Channels to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.41 HAL\_TIM\_PWM\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</code>
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 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</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>• HAL status</li> </ul>

#### 42.2.42 HAL\_TIM\_PWM\_Stop\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
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 TIM_HandleTypeDef 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.43 HAL\_TIM\_IC\_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)</code>
---------------	---

Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the <b>TIM_HandleTypeDef</b> and create the associated handle.
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>• HAL status</li> </ul>

#### 42.2.44 HAL\_TIM\_IC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes the TIM peripheral.
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>• HAL status</li> </ul>

#### 42.2.45 HAL\_TIM\_IC\_MspInit

Function Name	<b>void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Input Capture MSP.
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>• None</li> </ul>

#### 42.2.46 HAL\_TIM\_IC\_MspDeInit

Function Name	<b>void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Input Capture MSP.
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>• None</li> </ul>

#### 42.2.47 HAL\_TIM\_IC\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * 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 <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: <b>TIM_CHANNEL_1:</b> TIM Channel 1 selected<b>TIM_CHANNEL_2:</b> TIM Channel 2 selected<b>TIM_CHANNEL_3:</b> TIM Channel 3 selected<b>TIM_CHANNEL_4:</b> TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.48 HAL\_TIM\_IC\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops 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 disabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 42.2.49 HAL\_TIM\_IC\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</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 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 42.2.50 HAL\_TIM\_IC\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</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 TIM_HandleTypeDef 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 42.2.51 HAL\_TIM\_IC\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</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 TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li><b>Channel:</b> TIM Channels to be enabled. This parameter can</li> </ul>

be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

#### Return values

- HAL status

### 42.2.52 HAL\_TIM\_IC\_Stop\_DMA

#### Function Name

**HAL\_StatusTypeDef HAL\_TIM\_IC\_Stop\_DMA  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)**

#### Function Description

Stops the TIM Input Capture measurement on in DMA mode.

#### Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channels to be disabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

#### Return values

- HAL status

### 42.2.53 HAL\_TIM\_OnePulse\_Init

#### Function Name

**HAL\_StatusTypeDef HAL\_TIM\_OnePulse\_Init  
(TIM\_HandleTypeDef \* htim, uint32\_t OnePulseMode)**

#### Function Description

Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM\_HandleTypeDef and create the associated handle.

#### Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values: TIM\_OPMODE\_SINGLE: Only one pulse will be generated.TIM\_OPMODE\_REPETITIVE: Repetitive pulses will be generated.

#### Return values

- HAL status

### 42.2.54 HAL\_TIM\_OnePulse\_DelInit

#### Function Name

**HAL\_StatusTypeDef HAL\_TIM\_OnePulse\_DelInit  
(TIM\_HandleTypeDef \* htim)**

#### Function Description

Deinitializes the TIM One Pulse.

#### Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

#### Return values

- HAL status

### 42.2.55 HAL\_TIM\_OnePulse\_MspInit

Function Name	<b>void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM One Pulse 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>

#### 42.2.56 HAL\_TIM\_OnePulse\_MspDeInit

Function Name	<b>void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM One Pulse 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>

#### 42.2.57 HAL\_TIM\_OnePulse\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * 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 TIM_HandleTypeDef 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.58 HAL\_TIM\_OnePulse\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * 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 TIM_HandleTypeDef 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.59 HAL\_TIM\_OnePulse\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * 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 TIM_HandleTypeDef structure that</li> </ul>

- contains the configuration information for TIM module.
- **OutputChannel:** : TIM Channels to be enabled. This parameter can be one of the following values:  
TIM\_CHANNEL\_1: TIM Channel 1  
selectedTIM\_CHANNEL\_2: TIM Channel 2 selected

Return values

- HAL status

#### 42.2.60 HAL\_TIM\_OnePulse\_Stop\_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</code>
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 TIM_HandleTypeDef 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	• HAL status

#### 42.2.61 HAL\_TIM\_Encoder\_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)</code>
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 TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>sConfig:</b> TIM Encoder Interface configuration structure</li> </ul>
Return values	• HAL status

#### 42.2.62 HAL\_TIM\_Encoder\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_TIM_Encoder_DelInit (TIM_HandleTypeDef * htim)</code>
Function Description	Deinitializes the TIM Encoder 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	• HAL status

#### 42.2.63 HAL\_TIM\_Encoder\_MspInit

Function Name	<code>void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM Encoder Interface 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>
<b>42.2.64 HAL_TIM_Encoder_MspDeInit</b>	
Function Name	<b>void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes TIM Encoder Interface 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>
<b>42.2.65 HAL_TIM_Encoder_Start</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * 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 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
<b>42.2.66 HAL_TIM_Encoder_Stop</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * 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 TIM_HandleTypeDef 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
<b>42.2.67 HAL_TIM_Encoder_Start_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</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 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2</li> </ul>

selectedTIM\_CHANNEL\_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- HAL status

#### 42.2.68 HAL\_TIM\_Encoder\_Stop\_IT

Function Name

**HAL\_StatusTypeDef HAL\_TIM\_Encoder\_Stop\_IT  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)**

Function Description

Stops the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channels to be disabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- HAL status

#### 42.2.69 HAL\_TIM\_Encoder\_Start\_DMA

Function Name

**HAL\_StatusTypeDef HAL\_TIM\_Encoder\_Start\_DMA  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel, uint32\_t \*  
pData1, uint32\_t \* pData2, uint16\_t Length)**

Function Description

Starts the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_ALL: TIM Channel 1 and TIM Channel 2 are selected
- **pData1:** The destination Buffer address for IC1.
- **pData2:** The destination Buffer address for IC2.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- HAL status

#### 42.2.70 HAL\_TIM\_Encoder\_Stop\_DMA

Function Name

**HAL\_StatusTypeDef HAL\_TIM\_Encoder\_Stop\_DMA  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)**

Function Description

Stops the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_ALL: TIM Channel 1 and TIM Channel 2 are selected

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

#### 42.2.71 HAL\_TIM\_IRQHandler

Function Name      **void HAL\_TIM\_IRQHandler (TIM\_HandleTypeDef \* htim)**

Function Description      This function handles TIM interrupts requests.

Parameters     
 

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values     
 

- None

#### 42.2.72 HAL\_TIM\_OC\_ConfigChannel

Function Name      **HAL\_StatusTypeDef HAL\_TIM\_OC\_ConfigChannel (TIM\_HandleTypeDef \* htim, TIM\_OC\_InitTypeDef \* sConfig, uint32\_t Channel)**

Function Description      Initializes the TIM Output Compare Channels according to the specified parameters in the TIM\_OC\_InitTypeDef.

Parameters     
 

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **sConfig:** TIM Output Compare configuration structure
- **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values     
 

- HAL status

#### 42.2.73 HAL\_TIM\_IC\_ConfigChannel

Function Name      **HAL\_StatusTypeDef HAL\_TIM\_IC\_ConfigChannel (TIM\_HandleTypeDef \* htim, TIM\_IC\_InitTypeDef \* sConfig, uint32\_t Channel)**

Function Description      Initializes the TIM Input Capture Channels according to the specified parameters in the TIM\_IC\_InitTypeDef.

Parameters     
 

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **sConfig:** TIM Input Capture configuration structure
- **Channel:** TIM Channels to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values     
 

- HAL status

#### 42.2.74 HAL\_TIM\_PWM\_ConfigChannel

Function Name      **HAL\_StatusTypeDef HAL\_TIM\_PWM\_ConfigChannel (TIM\_HandleTypeDef \* htim, TIM\_OC\_InitTypeDef \* sConfig, uint32\_t Channel)**

Function Description	Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<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>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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 42.2.75 HAL\_TIM\_OnePulse\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)</b>
Function Description	Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.
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>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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> <li><b>InputChannel:</b> TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 42.2.76 HAL\_TIM\_DMABurst\_WriteStart

Fu   **HAL\_StatusTypeDef HAL\_TIM\_DMABurst\_WriteStart (TIM\_HandleTypeDef \* htim, uint32\_t BurstBaseAddress, uint32\_t BurstRequestSrc, uint32\_t \* BurstBuffer, uint32\_t BurstLength)**

n  
N  
a  
m  
e

Fu   Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

nc  
tio  
n  
D  
es  
cri  
pti  
on

Pa ra m et er s	<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>BurstBaseAddress:</b> TIM Base address from when the DMA will starts the Data write. This parameters can be on of the following values: TIM_DMABASE_CR1TIM_DMABASE_CR2TIM_DMABASE_SMCRTIM_DMABASE_DMABASE_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTIM_DMABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_CCERTIM_DMABASE_CNTTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_DMABASE_RCRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_BDTRTIM_DMABASE_DCR</li> <li>• <b>BurstRequestSrc:</b> TIM DMA Request sources. This parameters can be on of the following values: TIM_DMA_UPDATE: TIM update Interrupt sourceTIM_DMA_CC1: TIM Capture Compare 1 DMA sourceTIM_DMA_CC2: TIM Capture Compare 2 DMA sourceTIM_DMA_CC3: TIM Capture Compare 3 DMA sourceTIM_DMA_CC4: TIM Capture Compare 4 DMA sourceTIM_DMA_COM: TIM Commutation DMA sourceTIM_DMA_TRIGGER: TIM Trigger DMA source</li> <li>• <b>BurstBuffer:</b> The Buffer address.</li> <li>• <b>BurstLength:</b> DMA Burst length. This parameter can be one value between TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.</li> </ul>
R et ur n va lu es	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.77 HAL\_TIM\_DMABurst\_WriteStop

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * 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 TIM_HandleTypeDef 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>• HAL status</li> </ul>

#### 42.2.78 HAL\_TIM\_DMABurst\_ReadStart

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t 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 TIM_HandleTypeDef 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 on of the following values:</li> </ul>

TIM\_DMABASE\_CR1TIM\_DMABASE\_CR2TIM\_DMABASE\_  
 SMCRTIM\_DMABASE\_DIERTIM\_DMABASE\_SRTIM\_DMAB  
 ASE\_EGRTIM\_DMABASE\_CCMR1TIM\_DMABASE\_CCMR2  
 TIM\_DMABASE\_CCERTIM\_DMABASE\_CNTTIM\_DMABASE  
 \_PSCTIM\_DMABASE\_ARRTIM\_DMABASE\_RCRTIM\_DMAB  
 ASE\_CCR1TIM\_DMABASE\_CCR2TIM\_DMABASE\_CCR3TI  
 M\_DMABASE\_CCR4TIM\_DMABASE\_BDTRTIM\_DMABASE\_

- **BurstRequestSrc:** TIM DMA Request sources. This parameters can be one of the following values:  
 TIM\_DMA\_UPDATE: TIM update Interrupt  
 sourceTIM\_DMA\_CC1: TIM Capture Compare 1 DMA  
 sourceTIM\_DMA\_CC2: TIM Capture Compare 2 DMA  
 sourceTIM\_DMA\_CC3: TIM Capture Compare 3 DMA  
 sourceTIM\_DMA\_CC4: TIM Capture Compare 4 DMA  
 sourceTIM\_DMA\_COM: TIM Commutation DMA  
 sourceTIM\_DMA\_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between TIM\_DMABURSTLENGTH\_1TRANSFER and TIM\_DMABURSTLENGTH\_18TRANSFERS.

## Return values

- HAL status

**42.2.79 HAL\_TIM\_DMABurst\_ReadStop**

Function Name      **HAL\_StatusTypeDef HAL\_TIM\_DMABurst\_ReadStop**  
**(TIM\_HandleTypeDef \* htim, uint32\_t BurstRequestSrc)**

Function Description      Stop the DMA burst reading.

- Parameters
- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
  - **BurstRequestSrc:** TIM DMA Request sources to disable.

## Return values

- HAL status

**42.2.80 HAL\_TIM\_GenerateEvent**

Function Name      **HAL\_StatusTypeDef HAL\_TIM\_GenerateEvent**  
**(TIM\_HandleTypeDef \* htim, uint32\_t EventSource)**

Function Description      Generate a software event.

- Parameters
- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
  - **EventSource:** specifies the event source. This parameter can be one of the following values:  
 TIM\_EVENTSOURCE\_UPDATE: Timer update Event  
 sourceTIM\_EVENTSOURCE\_CC1: Timer Capture Compare 1 Event  
 sourceTIM\_EVENTSOURCE\_CC2: Timer Capture Compare 2 Event  
 sourceTIM\_EVENTSOURCE\_CC3: Timer Capture Compare 3 Event  
 sourceTIM\_EVENTSOURCE\_CC4: Timer Capture Compare 4 Event  
 sourceTIM\_EVENTSOURCE\_COM: Timer COM event source  
 sourceTIM\_EVENTSOURCE\_TRIGGER: Timer Trigger Event source  
 sourceTIM\_EVENTSOURCE\_BREAK: Timer Break

	event source
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• TIM6 and TIM7 can only generate an update event.</li> <li>• TIM_EVENTSOURCE_COM and TIM_EVENTSOURCE_BREAK are used only with TIM1 and TIM8.</li> </ul>

#### 42.2.81 HAL\_TIM\_ConfigOCrefClear

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * 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 TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>sClearInputConfig:</b> pointer to a TIM_ClearInputConfigTypeDef 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.82 HAL\_TIM\_ConfigClockSource

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)</b>
Function Description	Configures the clock source to be used.
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>sClockSourceConfig:</b> pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.83 HAL\_TIM\_ConfigTI1Input

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)</b>
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 TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>TI1_Selection:</b> Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can</li> </ul>

be one of the following values: TIM\_TI1SELECTION\_CH1:  
 The TIMx\_CH1 pin is connected to TI1  
 inputTIM\_TI1SELECTION\_XORCOMBINATION: The  
 TIMx\_CH1, CH2 and CH3 pins are connected to the TI1 input  
 (XOR combination)

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

#### 42.2.84 HAL\_TIM\_SlaveConfigSynchronization

Function Name	<b>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)</b>
Function Description	Configures the TIM in Slave 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>sSlaveConfig:</b> pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the ) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.85 HAL\_TIM\_SlaveConfigSynchronization\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)</b>
Function Description	Configures the TIM in Slave mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> TIM handle.</li> <li>• <b>sSlaveConfig:</b> pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the ) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.86 HAL\_TIM\_ReadCapturedValue

Function Name	<b>uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)</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 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: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>

Return values	<ul style="list-style-type: none"> <li>Captured value</li> </ul>
---------------	--

#### 42.2.87 HAL\_TIM\_PeriodElapsedCallback

Function Name	<b>void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Period elapsed callback in non blocking 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>None</li> </ul>

#### 42.2.88 HAL\_TIM\_OC\_DelayElapsedCallback

Function Name	<b>void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Output Compare callback in non blocking 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>None</li> </ul>

#### 42.2.89 HAL\_TIM\_IC\_CaptureCallback

Function Name	<b>void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)</b>
Function Description	Input Capture callback in non blocking 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>None</li> </ul>

#### 42.2.90 HAL\_TIM\_PWM\_PulseFinishedCallback

Function Name	<b>void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)</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 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>

#### 42.2.91 HAL\_TIM\_TriggerCallback

Function Name	<b>void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)</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 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>

**42.2.92 HAL\_TIM\_ErrorCallback**

Function Name	<b>void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * 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 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>

**42.2.93 HAL\_TIM\_Base\_GetState**

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM Base 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>• HAL state</li></ul>

**42.2.94 HAL\_TIM\_OC\_GetState**

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM OC 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>• HAL state</li></ul>

**42.2.95 HAL\_TIM\_PWM\_GetState**

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM PWM 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>• HAL state</li></ul>

**42.2.96 HAL\_TIM\_IC\_GetState**

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM Input Capture 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>• HAL state</li></ul>

**42.2.97 HAL\_TIM\_OnePulse\_GetState**

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)</b>
Function Description	Return the TIM One Pulse 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>HAL state</li> </ul>

### 42.2.98 HAL\_TIM\_Encoder\_GetState

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)</b>
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>HAL state</li> </ul>

## 42.3 TIM Firmware driver defines

### 42.3.1 TIM

#### **TIM AOE Bit State**

`TIM_AUTOMATICOUTPUT_ENABLE`  
`TIM_AUTOMATICOUTPUT_DISABLE`

#### **TIM Break Input State**

`TIM_BREAK_ENABLE`  
`TIM_BREAK_DISABLE`

#### **TIM Break Polarity**

`TIM_BREAKPOLARITY_LOW`  
`TIM_BREAKPOLARITY_HIGH`

#### **TIM Channel**

`TIM_CHANNEL_1`  
`TIM_CHANNEL_2`  
`TIM_CHANNEL_3`  
`TIM_CHANNEL_4`  
`TIM_CHANNEL_ALL`

#### **TIM Clear Input Polarity**

`TIM_CLEARINPUTPOLARITY_INVERTED`      Polarity for ETRx pin  
`TIM_CLEARINPUTPOLARITY_NONINVERTED`      Polarity for ETRx pin

#### **TIM Clear Input Prescaler**

`TIM_CLEARINPUTPRESCALER_DIV1`      No prescaler is used  
`TIM_CLEARINPUTPRESCALER_DIV2`      Prescaler for External ETR pin: Capture

TIM_CLEARINPUTPRESCALER_DIV4	Prescaler for External ETR pin: Capture performed once every 4 events.
TIM_CLEARINPUTPRESCALER_DIV8	Prescaler for External ETR pin: Capture performed once every 8 events.

***TIM Clear Input Source***

TIM\_CLEARINPUTSOURCE\_ETR  
TIM\_CLEARINPUTSOURCE\_NONE

***TIM Clock Division***

TIM\_CLOCKDIVISION\_DIV1  
TIM\_CLOCKDIVISION\_DIV2  
TIM\_CLOCKDIVISION\_DIV4

***TIM Clock Polarity***

TIM_CLOCKPOLARITY_INVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_NONINVERTED	Polarity for ETRx clock sources
TIM_CLOCKPOLARITY_RISING	Polarity for TIx clock sources
TIM_CLOCKPOLARITY_FALLING	Polarity for TIx clock sources
TIM_CLOCKPOLARITY_BOTHEDGE	Polarity for TIx clock sources

***TIM Clock Prescaler***

TIM_CLOCKPRESCALER_DIV1	No prescaler is used
TIM_CLOCKPRESCALER_DIV2	Prescaler for External ETR Clock: Capture performed once every 2 events.
TIM_CLOCKPRESCALER_DIV4	Prescaler for External ETR Clock: Capture performed once every 4 events.
TIM_CLOCKPRESCALER_DIV8	Prescaler for External ETR Clock: Capture performed once every 8 events.

***TIM Clock Source***

TIM\_CLOCKSOURCE\_ETRMODE2  
TIM\_CLOCKSOURCE\_INTERNAL  
TIM\_CLOCKSOURCE\_ITR0  
TIM\_CLOCKSOURCE\_ITR1  
TIM\_CLOCKSOURCE\_ITR2  
TIM\_CLOCKSOURCE\_ITR3  
TIM\_CLOCKSOURCE\_TI1ED  
TIM\_CLOCKSOURCE\_TI1  
TIM\_CLOCKSOURCE\_TI2  
TIM\_CLOCKSOURCE\_ETRMODE1

***TIM Commutation Source***

TIM\_COMMUTATION\_TRGI  
TIM\_COMMUTATION\_SOFTWARE  
**TIM Counter Mode**  
TIM\_COUNTERMODE\_UP  
TIM\_COUNTERMODE\_DOWN  
TIM\_COUNTERMODE\_CENTERALIGNED1  
TIM\_COUNTERMODE\_CENTERALIGNED2  
TIM\_COUNTERMODE\_CENTERALIGNED3  
**TIM DMA Base address**  
TIM\_DMABASE\_CR1  
TIM\_DMABASE\_CR2  
TIM\_DMABASE\_SMCR  
TIM\_DMABASE\_DIER  
TIM\_DMABASE\_SR  
TIM\_DMABASE\_EGR  
TIM\_DMABASE\_CCMR1  
TIM\_DMABASE\_CCMR2  
TIM\_DMABASE\_CCER  
TIM\_DMABASE\_CNT  
TIM\_DMABASE\_PSC  
TIM\_DMABASE\_ARR  
TIM\_DMABASE\_RCR  
TIM\_DMABASE\_CCR1  
TIM\_DMABASE\_CCR2  
TIM\_DMABASE\_CCR3  
TIM\_DMABASE\_CCR4  
TIM\_DMABASE\_BDTR  
TIM\_DMABASE\_DCR  
TIM\_DMABASE\_OR  
**TIM DMA Burst Length**  
TIM\_DMABURSTLENGTH\_1TRANSFER  
TIM\_DMABURSTLENGTH\_2TRANSFERS  
TIM\_DMABURSTLENGTH\_3TRANSFERS  
TIM\_DMABURSTLENGTH\_4TRANSFERS  
TIM\_DMABURSTLENGTH\_5TRANSFERS  
TIM\_DMABURSTLENGTH\_6TRANSFERS

TIM\_DMABURSTLENGTH\_7TRANSFERS  
TIM\_DMABURSTLENGTH\_8TRANSFERS  
TIM\_DMABURSTLENGTH\_9TRANSFERS  
TIM\_DMABURSTLENGTH\_10TRANSFERS  
TIM\_DMABURSTLENGTH\_11TRANSFERS  
TIM\_DMABURSTLENGTH\_12TRANSFERS  
TIM\_DMABURSTLENGTH\_13TRANSFERS  
TIM\_DMABURSTLENGTH\_14TRANSFERS  
TIM\_DMABURSTLENGTH\_15TRANSFERS  
TIM\_DMABURSTLENGTH\_16TRANSFERS  
TIM\_DMABURSTLENGTH\_17TRANSFERS  
TIM\_DMABURSTLENGTH\_18TRANSFERS

***TIM DMA sources***

TIM\_DMA\_UPDATE  
TIM\_DMA\_CC1  
TIM\_DMA\_CC2  
TIM\_DMA\_CC3  
TIM\_DMA\_CC4  
TIM\_DMA\_COM  
TIM\_DMA\_TRIGGER

***TIM Encoder Mode***

TIM\_ENCODERMODE\_TI1  
TIM\_ENCODERMODE\_TI2  
TIM\_ENCODERMODE\_TI12

***TIM ETR Polarity***

TIM\_ETRPOLARITY\_INVERTED      Polarity for ETR source  
TIM\_ETRPOLARITY\_NONINVERTED    Polarity for ETR source

***TIM ETR Prescaler***

TIM\_ETRPRESCALER\_DIV1    No prescaler is used  
TIM\_ETRPRESCALER\_DIV2    ETR input source is divided by 2  
TIM\_ETRPRESCALER\_DIV4    ETR input source is divided by 4  
TIM\_ETRPRESCALER\_DIV8    ETR input source is divided by 8

***TIM Event Source***

TIM\_EVENTSOURCE\_UPDATE  
TIM\_EVENTSOURCE\_CC1  
TIM\_EVENTSOURCE\_CC2

TIM\_EVENTSOURCE\_CC3  
TIM\_EVENTSOURCE\_CC4  
TIM\_EVENTSOURCE\_COM  
TIM\_EVENTSOURCE\_TRIGGER  
TIM\_EVENTSOURCE\_BREAK

***TIM Exported Macros***

<code>_HAL_TIM_RESET_HANDLE_ST</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Reset TIM handle state.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: TIM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>_HAL_TIM_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Enable the TIM peripheral.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: TIM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>_HAL_TIM_MOE_ENABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Enable the TIM main Output.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: TIM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>_HAL_TIM_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Disable the TIM peripheral.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: TIM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>
<code>_HAL_TIM_MOE_DISABLE</code>	<b>Description:</b> <ul style="list-style-type: none"><li>Disable the TIM main Output.</li></ul> <b>Parameters:</b> <ul style="list-style-type: none"><li><code>_HANDLE_</code>: TIM handle</li></ul> <b>Return value:</b> <ul style="list-style-type: none"><li>None</li></ul>

```
__HAL_TIM_ENABLE_IT  
__HAL_TIM_ENABLE_DMA  
__HAL_TIM_DISABLE_IT  
__HAL_TIM_DISABLE_DMA  
__HAL_TIM_GET_FLAG  
__HAL_TIM_CLEAR_FLAG  
__HAL_TIM_GET_IT_SOURCE  
__HAL_TIM_CLEAR_IT  
__HAL_TIM_IS_TIM_COUNTING_  
DOWN  
__HAL_TIM_SET_PRESCALER  
TIM_SET_ICPRESCALERVALUE  
TIM_RESET_ICPRESCALERVAL  
UE  
TIM_SET_CAPTUREPOLARITY  
TIM_RESET_CAPTUREPOLARIT  
Y  
__HAL_TIM_SET_COMPARE
```

**Description:**

- Sets the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.
- \_\_CHANNEL\_\_: : TIM Channels to be configured. This parameter can be one of the following values:
  - TIM\_CHANNEL\_1: TIM Channel 1 selected
  - TIM\_CHANNEL\_2: TIM Channel 2 selected
  - TIM\_CHANNEL\_3: TIM Channel 3 selected
  - TIM\_CHANNEL\_4: TIM Channel 4 selected
- \_\_COMPARE\_\_: specifies the Capture Compare register new value.

**Return value:**

- None

```
__HAL_TIM_GET_COMPARE
```

**Description:**

- Gets the TIM Capture Compare Register value on runtime.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.

- \_\_CHANNEL\_\_: : TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM\_CHANNEL\_1: get capture/compare 1 register value
  - TIM\_CHANNEL\_2: get capture/compare 2 register value
  - TIM\_CHANNEL\_3: get capture/compare 3 register value
  - TIM\_CHANNEL\_4: get capture/compare 4 register value

**Return value:**

- None

\_\_HAL\_TIM\_SET\_COUNTER

**Description:**

- Sets the TIM Counter Register value on runtime.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.
- \_\_COUNTER\_\_: specifies the Counter register new value.

**Return value:**

- None

\_\_HAL\_TIM\_GET\_COUNTER

**Description:**

- Gets the TIM Counter Register value on runtime.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.

**Return value:**

- None

\_\_HAL\_TIM\_SET\_AUTORELOAD

**Description:**

- Sets the TIM Autoreload Register value on runtime without calling another time any Init function.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.
- \_\_AUTORELOAD\_\_: specifies the Counter register new value.

**Return value:**

- None

\_\_HAL\_TIM\_GET\_AUTORELOAD

**Description:**

- Gets the TIM Autoreload Register value on runtime.

**Parameters:**

- `_HANDLE_`: TIM handle.

**Return value:**

- None

`_HAL_TIM_SET_CLOCKDIVISION`  
N

**Description:**

- Sets the TIM Clock Division value on runtime without calling another time any Init function.

**Parameters:**

- `_HANDLE_`: TIM handle.
- `_CKD_`: specifies the clock division value. This parameter can be one of the following value:
  - `TIM_CLOCKDIVISION_DIV1`
  - `TIM_CLOCKDIVISION_DIV2`
  - `TIM_CLOCKDIVISION_DIV4`

**Return value:**

- None

`_HAL_TIM_GET_CLOCKDIVISION`  
N

**Description:**

- Gets the TIM Clock Division value on runtime.

**Parameters:**

- `_HANDLE_`: TIM handle.

**Return value:**

- None

`_HAL_TIM_SET_ICPRESCALER`

**Description:**

- Sets the TIM Input Capture prescaler on runtime without calling another time

**Parameters:**

- `_HANDLE_`: TIM handle.
- `_CHANNEL_`: : TIM Channels to be configured. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: TIM Channel 1 selected
  - `TIM_CHANNEL_2`: TIM Channel 2 selected
  - `TIM_CHANNEL_3`: TIM Channel 3 selected
  - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `_ICPSC_`: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  - `TIM_ICPSC_DIV1`: no prescaler
  - `TIM_ICPSC_DIV2`: capture is done once

- every 2 events
- TIM\_ICPSC\_DIV4: capture is done once every 4 events
- TIM\_ICPSC\_DIV8: capture is done once every 8 events

**Return value:**

- None

[\\_\\_HAL\\_TIM\\_GET\\_ICPRESCALER](#)**Description:**

- Gets the TIM Input Capture prescaler on runtime.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): TIM handle.
- [\\_\\_CHANNEL\\_\\_](#): TIM Channels to be configured. This parameter can be one of the following values:
  - TIM\_CHANNEL\_1: get input capture 1 prescaler value
  - TIM\_CHANNEL\_2: get input capture 2 prescaler value
  - TIM\_CHANNEL\_3: get input capture 3 prescaler value
  - TIM\_CHANNEL\_4: get input capture 4 prescaler value

**Return value:**

- None

[\\_\\_HAL\\_TIM\\_URS\\_ENABLE](#)**Description:**

- Set the Update Request Source (URS) bit of the TIMx\_CR1 register.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): TIM handle.

**Return value:**

- None

**Notes:**

- When the USR bit of the TIMx\_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

[\\_\\_HAL\\_TIM\\_URS\\_DISABLE](#)**Description:**

- Reset the Update Request Source (URS) bit of the TIMx\_CR1 register.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): TIM handle.

**Return value:**

- None

**Notes:**

- When the USR bit of the TIMx\_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled):
  - \_ Counter overflow/underflow
  - \_ Setting the UG bit
  - \_ Update generation through the slave mode controller

`__HAL_TIM_SET_CAPTUREPOLARITY`

**Description:**

- Sets the TIM Capture x input polarity on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: TIM Channel 1 selected
  - `TIM_CHANNEL_2`: TIM Channel 2 selected
  - `TIM_CHANNEL_3`: TIM Channel 3 selected
  - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__POLARITY__`: Polarity for TIx source
  - `TIM_INPUTCHANNELPOLARITY_RISING`: Rising Edge
  - `TIM_INPUTCHANNELPOLARITY_FALLING`: Falling Edge
  - `TIM_INPUTCHANNELPOLARITY_BOTHEDGE`: Rising and Falling Edge

**Return value:**

- None

**Notes:**

- The polarity `TIM_INPUTCHANNELPOLARITY_BOTHEDGE` is not authorized for TIM Channel 4.

***TIM Flag definition***

`TIM_FLAG_UPDATE`  
`TIM_FLAG_CC1`  
`TIM_FLAG_CC2`  
`TIM_FLAG_CC3`  
`TIM_FLAG_CC4`  
`TIM_FLAG_COM`  
`TIM_FLAG_TRIGGER`

TIM\_FLAG\_BREAK  
 TIM\_FLAG\_CC1OF  
 TIM\_FLAG\_CC2OF  
 TIM\_FLAG\_CC3OF  
 TIM\_FLAG\_CC4OF

***TIM Input Capture Polarity***

TIM\_ICPOLARITY\_RISING  
 TIM\_ICPOLARITY\_FALLING  
 TIM\_ICPOLARITY\_BOTHEDGE

***TIM Input Capture Prescaler***

TIM_ICPSC_DIV1	Capture performed each time an edge is detected on the capture input
TIM_ICPSC_DIV2	Capture performed once every 2 events
TIM_ICPSC_DIV4	Capture performed once every 4 events
TIM_ICPSC_DIV8	Capture performed once every 8 events

***TIM Input Capture Selection***

TIM_ICSELECTION_DIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively
TIM_ICSELECTION_INDIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively
TIM_ICSELECTION_TRC	TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

***TIM Input Channel Polarity***

TIM_INPUTCHANNELPOLARITY_RISING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for TIx source

***TIM Interrupt definition***

TIM\_IT\_UPDATE  
 TIM\_IT\_CC1  
 TIM\_IT\_CC2  
 TIM\_IT\_CC3  
 TIM\_IT\_CC4  
 TIM\_IT\_COM  
 TIM\_IT\_TRIGGER  
 TIM\_IT\_BREAK

***TIM Private macros to check input parameters***

IS\_TIM\_COUNTER\_MODE  
 IS\_TIM\_CLOCKDIVISION\_DIV

IS\_TIM\_PWM\_MODE  
IS\_TIM\_OC\_MODE  
IS\_TIM\_FAST\_STATE  
IS\_TIM\_OC\_POLARITY  
IS\_TIM\_OCN\_POLARITY  
IS\_TIM\_OCIDLE\_STATE  
IS\_TIM\_OCNIDLE\_STATE  
IS\_TIM\_CHANNELS  
IS\_TIM\_OPM\_CHANNELS  
IS\_TIM\_COMPLEMENTARY\_CHANNELS  
IS\_TIM\_IC\_POLARITY  
IS\_TIM\_IC\_SELECTION  
IS\_TIM\_IC\_PRESCALER  
IS\_TIM\_OPM\_MODE  
IS\_TIM\_DMA\_SOURCE  
IS\_TIM\_ENCODER\_MODE  
IS\_TIM\_EVENT\_SOURCE  
IS\_TIM\_CLOCKSOURCE  
IS\_TIM\_CLOCKPOLARITY  
IS\_TIM\_CLOCKPRESCALER  
IS\_TIM\_CLOCKFILTER  
IS\_TIM\_CLEARINPUT\_SOURCE  
IS\_TIM\_CLEARINPUT\_POLARITY  
IS\_TIM\_CLEARINPUT\_PRESCALER  
IS\_TIM\_CLEARINPUT\_FILTER  
IS\_TIM\_OSSR\_STATE  
IS\_TIM\_OSSI\_STATE  
IS\_TIM\_LOCK\_LEVEL  
IS\_TIM\_BREAK\_STATE  
IS\_TIM\_BREAK\_POLARITY  
IS\_TIM\_AUTOMATIC\_OUTPUT\_STATE  
IS\_TIM\_TRGO\_SOURCE  
IS\_TIM\_SLAVE\_MODE  
IS\_TIM\_MSM\_STATE  
IS\_TIM\_TRIGGER\_SELECTION  
IS\_TIM\_INTERNAL\_TRIGGEREVENT\_SELECTION

IS\_TIM\_TRIGGERPOLARITY  
IS\_TIM\_TRIGGERPRESCALER  
IS\_TIM\_TRIGGERFILTER  
IS\_TIM\_TI1SELECTION  
IS\_TIM\_DMA\_BASE  
IS\_TIM\_DMA\_LENGTH  
IS\_TIM\_IC\_FILTER

***TIM Lock level***

TIM\_LOCKLEVEL\_OFF  
TIM\_LOCKLEVEL\_1  
TIM\_LOCKLEVEL\_2  
TIM\_LOCKLEVEL\_3

***TIM Mask Definition***

TIM\_CCER\_CCxE\_MASK  
TIM\_CCER\_CCxNE\_MASK

***TIM Master Mode Selection***

TIM\_TRGO\_RESET  
TIM\_TRGO\_ENABLE  
TIM\_TRGO\_UPDATE  
TIM\_TRGO\_OC1  
TIM\_TRGO\_OC1REF  
TIM\_TRGO\_OC2REF  
TIM\_TRGO\_OC3REF  
TIM\_TRGO\_OC4REF

***TIM Master Slave Mode***

TIM\_MASTERSLAVEMODE\_ENABLE  
TIM\_MASTERSLAVEMODE\_DISABLE

***TIM One Pulse Mode***

TIM\_OPMODE\_SINGLE  
TIM\_OPMODE\_REPETITIVE

***TIM OSSI OffState Selection for Idle mode state***

TIM\_OSSI\_ENABLE  
TIM\_OSSI\_DISABLE

***TIM OSSR OffState Selection for Run mode state***

TIM\_OSSR\_ENABLE  
TIM\_OSSR\_DISABLE

***TIM Output Compare and PWM modes***

TIM\_OCMODE\_TIMING

TIM\_OCMODE\_ACTIVE

TIM\_OCMODE\_INACTIVE

TIM\_OCMODE\_TOGGLE

TIM\_OCMODE\_PWM1

TIM\_OCMODE\_PWM2

TIM\_OCMODE\_FORCED\_ACTIVE

TIM\_OCMODE\_FORCED\_INACTIVE

***TIM Output Compare Idle State***

TIM\_OCIDLESTATE\_SET

TIM\_OCIDLESTATE\_RESET

***TIM Output Compare N Idle State***

TIM\_OCNIDLESTATE\_SET

TIM\_OCNIDLESTATE\_RESET

***TIM Output CompareN Polarity***

TIM\_OCPOLARITY\_HIGH

TIM\_OCPOLARITY\_LOW

***TIM Output Compare Polarity***

TIM\_OCPOLARITY\_HIGH

TIM\_OCPOLARITY\_LOW

***TIM Output Fast State***

TIM\_OCFAST\_DISABLE

TIM\_OCFAST\_ENABLE

***TIM Slave Mode***

TIM\_SLAVEMODE\_DISABLE

TIM\_SLAVEMODE\_RESET

TIM\_SLAVEMODE\_GATED

TIM\_SLAVEMODE\_TRIGGER

TIM\_SLAVEMODE\_EXTERNAL1

***TIM TI1 Selection***

TIM\_TI1SELECTION\_CH1

TIM\_TI1SELECTION\_XORCOMBINATION

***TIM Trigger Polarity***

TIM_TRIGGERPOLARITY_INVERTED	Polarity for ETRx trigger sources
TIM_TRIGGERPOLARITY_NONINVERTED	Polarity for ETRx trigger sources
TIM_TRIGGERPOLARITY_RISING	Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING	Polarity for TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_BOTHEDGE	Polarity for TIxFPx or TI1_ED trigger sources

***TIM Trigger Prescaler***

TIM_TRIGGERPRESCALER_DIV1	No prescaler is used
TIM_TRIGGERPRESCALER_DIV2	Prescaler for External ETR Trigger: Capture performed once every 2 events.
TIM_TRIGGERPRESCALER_DIV4	Prescaler for External ETR Trigger: Capture performed once every 4 events.
TIM_TRIGGERPRESCALER_DIV8	Prescaler for External ETR Trigger: Capture performed once every 8 events.

***TIM Trigger Selection***

TIM_TS_ITR0
TIM_TS_ITR1
TIM_TS_ITR2
TIM_TS_ITR3
TIM_TS_TI1F_ED
TIM_TS_TI1FP1
TIM_TS_TI2FP2
TIM_TS_ETRF
TIM_TS_NONE

## 43 HAL TIM Extension Driver

### 43.1 TIME Firmware driver registers structures

#### 43.1.1 TIM\_HallSensor\_InitTypeDef

##### Data Fields

- *uint32\_t IC1Polarity*
- *uint32\_t IC1Prescaler*
- *uint32\_t IC1Filter*
- *uint32\_t Commutation\_Delay*

##### Field Documentation

- *uint32\_t TIM\_HallSensor\_InitTypeDef::IC1Polarity*  
Specifies the active edge of the input signal. This parameter can be a value of [\*TIM\\_Input\\_Capture\\_Polarity\*](#)
- *uint32\_t TIM\_HallSensor\_InitTypeDef::IC1Prescaler*  
Specifies the Input Capture Prescaler. This parameter can be a value of [\*TIM\\_Input\\_Capture\\_Prescaler\*](#)
- *uint32\_t TIM\_HallSensor\_InitTypeDef::IC1Filter*  
Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF
- *uint32\_t TIM\_HallSensor\_InitTypeDef::Commutation\_Delay*  
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF

#### 43.1.2 TIM\_MasterConfigTypeDef

##### Data Fields

- *uint32\_t MasterOutputTrigger*
- *uint32\_t MasterSlaveMode*

##### Field Documentation

- *uint32\_t TIM\_MasterConfigTypeDef::MasterOutputTrigger*  
Trigger output (TRGO) selection. This parameter can be a value of [\*TIM\\_Master\\_Mode\\_Selection\*](#)
- *uint32\_t TIM\_MasterConfigTypeDef::MasterSlaveMode*  
Master/slave mode selection. This parameter can be a value of [\*TIM\\_Master\\_Slave\\_Mode\*](#)

#### 43.1.3 TIM\_BreakDeadTimeConfigTypeDef

**Data Fields**

- *uint32\_t OffStateRunMode*
- *uint32\_t OffStateIDLEMode*
- *uint32\_t LockLevel*
- *uint32\_t DeadTime*
- *uint32\_t BreakState*
- *uint32\_t BreakPolarity*
- *uint32\_t AutomaticOutput*

**Field Documentation**

- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::OffStateRunMode*  
TIM off state in run mode. This parameter can be a value of  
*TIM\_OSSR\_Off\_State\_Selection\_for\_Run\_mode\_state*
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::OffStateIDLEMode*  
TIM off state in IDLE mode. This parameter can be a value of  
*TIM\_OSSI\_Off\_State\_Selection\_for\_Idle\_mode\_state*
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::LockLevel*  
TIM Lock level. This parameter can be a value of *TIM\_Lock\_level*
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::DeadTime*  
TIM dead Time. This parameter can be a number between Min\_Data = 0x00 and Max\_Data = 0xFF
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::BreakState*  
TIM Break State. This parameter can be a value of  
*TIM\_Break\_Input\_enable\_disable*
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::BreakPolarity*  
TIM Break input polarity. This parameter can be a value of *TIM\_Break\_Polarity*
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::AutomaticOutput*  
TIM Automatic Output Enable state. This parameter can be a value of  
*TIM\_AOE\_Bit\_Set\_Reset*

## 43.2 TIME Firmware driver API description

### 43.2.1 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

### 43.2.2 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().

### 43.2.3 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.

This section contains the following APIs:

- [\*\*\*HAL\\_TIMEx\\_HallSensor\\_Init\(\)\*\*\*](#)
- [\*\*\*HAL\\_TIMEx\\_HallSensor\\_DeInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_TIMEx\\_HallSensor\\_MspInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_TIMEx\\_HallSensor\\_MspDeInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_TIMEx\\_HallSensor\\_Start\(\)\*\*\*](#)
- [\*\*\*HAL\\_TIMEx\\_HallSensor\\_Stop\(\)\*\*\*](#)
- [\*\*\*HAL\\_TIMEx\\_HallSensor\\_Start\\_IT\(\)\*\*\*](#)
- [\*\*\*HAL\\_TIMEx\\_HallSensor\\_Stop\\_IT\(\)\*\*\*](#)

- [\*HAL\\_TIMEx\\_HallSensor\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIMEx\\_HallSensor\\_Stop\\_DMA\(\)\*](#)

#### 43.2.4 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.

This section contains the following APIs:

- [\*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\(\)\*](#)

#### 43.2.5 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.

This section contains the following APIs:

- [\*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\(\)\*](#)

#### 43.2.6 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.

This section contains the following APIs:

- [\*HAL\\_TIMEx\\_OnePulseN\\_Start\(\)\*](#)
- [\*HAL\\_TIMEx\\_OnePulseN\\_Stop\(\)\*](#)
- [\*HAL\\_TIMEx\\_OnePulseN\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIMEx\\_OnePulseN\\_Stop\\_IT\(\)\*](#)

### 43.2.7 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.

This section contains the following APIs:

- [\*HAL\\_TIMEx\\_ConfigCommuteEvent\(\)\*](#)
- [\*HAL\\_TIMEx\\_ConfigCommuteEvent\\_IT\(\)\*](#)
- [\*HAL\\_TIMEx\\_ConfigCommuteEvent\\_DMA\(\)\*](#)
- [\*HAL\\_TIMEx\\_MasterConfigSynchronization\(\)\*](#)
- [\*HAL\\_TIMEx\\_ConfigBreakDeadTime\(\)\*](#)
- [\*HAL\\_TIMEx\\_RemapConfig\(\)\*](#)

### 43.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- [\*HAL\\_TIMEx\\_CommuteCallback\(\)\*](#)
- [\*HAL\\_TIMEx\\_BreakCallback\(\)\*](#)
- [\*TIMEx\\_DMACommuteCplt\(\)\*](#)

### 43.2.9 Extension Peripheral State functions

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

This section contains the following APIs:

- [\*HAL\\_TIMEx\\_HallSensor\\_GetState\(\)\*](#)

### 43.2.10 HAL\_TIMEx\_HallSensor\_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)</code>
---------------	---

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 TIM_HandleTypeDef 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>HAL status</li> </ul>

### 43.2.11 HAL\_TIMEx\_HallSensor\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)</b>
Function Description	DeInitializes 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>HAL status</li> </ul>

### 43.2.12 HAL\_TIMEx\_HallSensor\_MspInit

Function Name	<b>void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)</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>

### 43.2.13 HAL\_TIMEx\_HallSensor\_MspDeInit

Function Name	<b>void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)</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>

### 43.2.14 HAL\_TIMEx\_HallSensor\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)</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>HAL status</li> </ul>

### 43.2.15 HAL\_TIMEx\_HallSensor\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)</b>
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>• HAL status</li> </ul>

### 43.2.16 HAL\_TIMEx\_HallSensor\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)</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>• HAL status</li> </ul>

### 43.2.17 HAL\_TIMEx\_HallSensor\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)</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>• HAL status</li> </ul>

### 43.2.18 HAL\_TIMEx\_HallSensor\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)</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 TIM_HandleTypeDef 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>• HAL status</li> </ul>

### 43.2.19 HAL\_TIMEx\_HallSensor\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)</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 TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 43.2.20 HAL\_TIMEx\_OCN\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</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 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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 43.2.21 HAL\_TIMEx\_OCN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)</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 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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 43.2.22 HAL\_TIMEx\_OCN\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</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 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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 43.2.23 HAL\_TIMEx\_OCN\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</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 TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>

- **Channel:** TIM Channel to be disabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

#### 43.2.24 HAL\_TIMEx\_OCN\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</code>
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 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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</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>• HAL status</li> </ul>

#### 43.2.25 HAL\_TIMEx\_OCN\_Stop\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 43.2.26 HAL\_TIMEx\_PWMN\_Start

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>

1 selectedTIM\_CHANNEL\_2: TIM Channel 2  
 selectedTIM\_CHANNEL\_3: TIM Channel 3  
 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values • HAL status

### 43.2.27 HAL\_TIMEx\_PWMN\_Stop

Function Name **HAL\_StatusTypeDef HAL\_TIMEx\_PWMN\_Stop(  
TIM\_HandleTypeDef \*htim, uint32\_t Channel)**

Function Description Stops the PWM signal generation on the complementary output.

Parameters • **htim**: pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.  
 • **Channel**: TIM Channel to be disabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values • HAL status

### 43.2.28 HAL\_TIMEx\_PWMN\_Start\_IT

Function Name **HAL\_StatusTypeDef HAL\_TIMEx\_PWMN\_Start\_IT(  
TIM\_HandleTypeDef \*htim, uint32\_t Channel)**

Function Description Starts the PWM signal generation in interrupt mode on the complementary output.

Parameters • **htim**: pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.  
 • **Channel**: TIM Channel to be disabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values • HAL status

### 43.2.29 HAL\_TIMEx\_PWMN\_Stop\_IT

Function Name **HAL\_StatusTypeDef HAL\_TIMEx\_PWMN\_Stop\_IT(  
TIM\_HandleTypeDef \*htim, uint32\_t Channel)**

Function Description Stops the PWM signal generation in interrupt mode on the complementary output.

Parameters • **htim**: pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.  
 • **Channel**: TIM Channel to be disabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values • HAL status

### 43.2.30 HAL\_TIMEx\_PWMN\_Start\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * 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 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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</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>• HAL status</li> </ul>

### 43.2.31 HAL\_TIMEx\_PWMN\_Stop\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * 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 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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 43.2.32 HAL\_TIMEx\_OnePulseN\_Start

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</code>
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: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 43.2.33 HAL\_TIMEx\_OnePulseN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
<b>43.2.34 HAL_TIMEx_OnePulseN_Start_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</b>
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 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: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
<b>43.2.35 HAL_TIMEx_OnePulseN_Stop_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</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 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 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
<b>43.2.36 HAL_TIMEx_ConfigCommutationEvent</b>	
Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)</b>
Function Description	Configure the TIM commutation event sequence.
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</li> </ul>

Interfacing with the Hall sensor. This parameter can be one of the following values: TIM\_TS\_ITR0: Internal trigger 0

selectedTIM\_TS\_ITR1: Internal trigger 1

selectedTIM\_TS\_ITR2: Internal trigger 2

selectedTIM\_TS\_ITR3: Internal trigger 3

selectedTIM\_TS\_NONE: No trigger is needed

- **CommutationSource:** the Commutation Event source. This parameter can be one of the following values:  
TIM\_COMMUTATION\_TRGI: Commutation source is the TRGI of the Interface  
TimerTIM\_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.

### 43.2.37 HAL\_TIMEx\_ConfigCommuteEvent\_IT

#### Function Name

**HAL\_StatusTypeDef**  
**HAL\_TIMEx\_ConfigCommuteEvent\_IT**  
(**TIM\_HandleTypeDef \* htim, uint32\_t InputTrigger, uint32\_t CommuteSource**)

#### Function Description

Configure the TIM commutation event sequence with interrupt.

#### Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values: TIM\_TS\_ITR0: Internal trigger 0  
selectedTIM\_TS\_ITR1: Internal trigger 1  
selectedTIM\_TS\_ITR2: Internal trigger 2  
selectedTIM\_TS\_ITR3: Internal trigger 3  
selectedTIM\_TS\_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source. This parameter can be one of the following values:  
TIM\_COMMUTATION\_TRGI: Commutation source is the TRGI of the Interface  
TimerTIM\_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.

### 43.2.38 HAL\_TIMEx\_ConfigCommutationEvent\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)</code>
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: Internal trigger 0 selectedTIM_TS_ITR1: Internal trigger 1 selectedTIM_TS_ITR2: Internal trigger 2 selectedTIM_TS_ITR3: Internal trigger 3 selectedTIM_TS_NONE: No trigger is needed</li> <li>• <b>CommutationSource:</b> the Commutation Event source. This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface TimerTIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</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>

### 43.2.39 HAL\_TIMEx\_MasterConfigSynchronization

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)</code>
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>• HAL status</li> </ul>

### 43.2.40 HAL\_TIMEx\_ConfigBreakDeadTime



Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)</b>
Function Description	Configures the Break feature, dead time, Lock level, OSSI/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_ConfigBreakDeadConfig_TypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 43.2.41 HAL\_TIMEx\_RemapConfig

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)</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> <li><b>Remap:</b> 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)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 43.2.42 HAL\_TIMEx\_CommmutationCallback

Function Name	<b>void HAL_TIMEx_CommmutationCallback (TIM_HandleTypeDef * htim)</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 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>

#### 43.2.43 HAL\_TIMEx\_BreakCallback

Function Name	<b>void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * 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 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>

#### 43.2.44 **TIMEx\_DMACommutationCplt**

Function Name	<b>void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)</b>
Function Description	TIM DMA Commutation 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>

#### 43.2.45 **HAL\_TIMEx\_HallSensor\_GetState**

Function Name	<b>HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (TIM_HandleTypeDef * 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 TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

### 43.3 **TIMEx Firmware driver defines**

#### 43.3.1 **TIMEx**

##### *TIM Private Macros*

IS\_TIM\_REMAP

IS\_TIM\_DEADTIME

##### *TIM Remap*

TIM\_TIM2\_TIM8\_TRGO

TIM\_TIM2\_ETH\_PTP

TIM\_TIM2\_USBFS\_SOF

TIM\_TIM2\_USBHS\_SOF

TIM\_TIM5\_GPIO

TIM\_TIM5\_LSI

TIM\_TIM5\_LSE

TIM\_TIM5\_RTC

TIM\_TIM11\_GPIO



## 44 HAL UART Generic Driver

### 44.1 UART Firmware driver registers structures

#### 44.1.1 UART\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t HwFlowCtl*
- *uint32\_t OverSampling*

##### Field Documentation

- ***uint32\_t UART\_InitTypeDef::BaudRate***  
This member configures the UART communication baud rate. The baud rate is computed using the following formula:  

$$\text{IntegerDivider} = ((\text{PCLKx}) / (8 * (\text{OVR8}+1)) * (\text{uart}->\text{Init}.BaudRate)))$$

$$\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{uint32\_t}) \text{IntegerDivider})) * 8 * (\text{OVR8}+1)) + 0.5$$

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\*\*](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32\_t UART\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [\*\*UART\\_Mode\*\*](#)
- ***uint32\_t UART\_InitTypeDef::HwFlowCtl***  
Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of [\*\*UART\\_Hardware\\_Flow\\_Control\*\*](#)
- ***uint32\_t UART\_InitTypeDef::OverSampling***  
Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of [\*\*UART\\_Over\\_Sampling\*\*](#)

#### 44.1.2 UART\_HandleTypeDef

##### Data Fields



- ***USART\_TypeDef \* Instance***
- ***UART\_HandleTypeDef Init***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferSize***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_UART\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***USART\_TypeDef\* UART\_HandleTypeDef::Instance***  
UART registers base address
- ***UART\_HandleTypeDef UART\_HandleTypeDef::Init***  
UART communication parameters
- ***uint8\_t\* UART\_HandleTypeDef::pTxBuffPtr***  
Pointer to UART Tx transfer Buffer
- ***uint16\_t UART\_HandleTypeDef::TxXferSize***  
UART Tx Transfer size
- ***uint16\_t UART\_HandleTypeDef::TxXferCount***  
UART Tx Transfer Counter
- ***uint8\_t\* UART\_HandleTypeDef::pRxBuffPtr***  
Pointer to UART Rx transfer Buffer
- ***uint16\_t UART\_HandleTypeDef::RxXferSize***  
UART Rx Transfer size
- ***uint16\_t UART\_HandleTypeDef::RxXferCount***  
UART Rx Transfer Counter
- ***DMA\_HandleTypeDef\* UART\_HandleTypeDef::hdmatx***  
UART Tx DMA Handle parameters
- ***DMA\_HandleTypeDef\* UART\_HandleTypeDef::hdmarx***  
UART Rx DMA Handle parameters
- ***HAL\_LockTypeDef UART\_HandleTypeDef::Lock***  
Locking object
- ***\_\_IO HAL\_UART\_StateTypeDef UART\_HandleTypeDef::State***  
UART communication state
- ***\_\_IO uint32\_t UART\_HandleTypeDef::ErrorCode***  
UART Error code

## 44.2 UART Firmware driver API description

### 44.2.1 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 transfer HAL\_UART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_UART\_Receive\_IT()

- At reception end of transfer HAL\_UART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxCpltCallback
- In case of transfer Error, HAL\_UART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_UART\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_UART\_Transmit\_DMA()
- At transmission end of half transfer HAL\_UART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxHalfCpltCallback
- At transmission end of transfer HAL\_UART\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_UART\_Receive\_DMA()
- At reception end of half transfer HAL\_UART\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxHalfCpltCallback
- At reception end of transfer HAL\_UART\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_UART\_RxCpltCallback
- In case of transfer Error, HAL\_UART\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_UART\_ErrorCallback
- Pause the DMA Transfer using HAL\_UART\_DMAPause()
- Resume the DMA Transfer using HAL\_UART\_DMAResume()
- Stop the DMA Transfer using HAL\_UART\_DMAStop()

### UART HAL driver macros list

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

- \_\_HAL\_UART\_ENABLE: Enable the UART peripheral
- \_\_HAL\_UART\_DISABLE: Disable the UART peripheral
- \_\_HAL\_UART\_GET\_FLAG : Check whether the specified UART flag is set or not
- \_\_HAL\_UART\_CLEAR\_FLAG : Clear the specified UART pending flag
- \_\_HAL\_UART\_ENABLE\_IT: Enable the specified UART interrupt
- \_\_HAL\_UART\_DISABLE\_IT: Disable the specified UART interrupt
- \_\_HAL\_UART\_GET\_IT\_SOURCE: Check whether the specified UART interrupt has occurred or not



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

#### 44.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit

- Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), please refer to Reference manual for possible UART frame formats.
- Hardware flow control
- Receiver/transmitter modes
- Over Sampling Method

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)).

This section contains the following APIs:

- [\*\*\*HAL\\_UART\\_Init\(\)\*\*\*](#)
- [\*\*\*HAL\\_HalfDuplex\\_Init\(\)\*\*\*](#)
- [\*\*\*HAL\\_LIN\\_Init\(\)\*\*\*](#)
- [\*\*\*HAL\\_MultiProcessor\\_Init\(\)\*\*\*](#)
- [\*\*\*HAL\\_UART\\_Delnit\(\)\*\*\*](#)
- [\*\*\*HAL\\_UART\\_MspInit\(\)\*\*\*](#)
- [\*\*\*HAL\\_UART\\_MspDelnit\(\)\*\*\*](#)

#### 44.2.3 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.

This section contains the following APIs:

- [\*HAL\\_UART\\_Transmit\(\)\*](#)
- [\*HAL\\_UART\\_Receive\(\)\*](#)
- [\*HAL\\_UART\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_UART\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_UART\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_UART\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_UART\\_DMAPause\(\)\*](#)
- [\*HAL\\_UART\\_DMAResume\(\)\*](#)
- [\*HAL\\_UART\\_DMADelete\(\)\*](#)
- [\*HAL\\_UART\\_IRQHandler\(\)\*](#)
- [\*HAL\\_UART\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_TxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_ErrorCallback\(\)\*](#)

#### 44.2.4 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.

This section contains the following APIs:

- [\*HAL\\_LIN\\_SendBreak\(\)\*](#)
- [\*HAL\\_MultiProcessor\\_EnterMuteMode\(\)\*](#)
- [\*HAL\\_MultiProcessor\\_ExitMuteMode\(\)\*](#)
- [\*HAL\\_HalfDuplex\\_EnableTransmitter\(\)\*](#)
- [\*HAL\\_HalfDuplex\\_EnableReceiver\(\)\*](#)

#### 44.2.5 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.

This section contains the following APIs:

- [\*HAL\\_UART\\_GetState\(\)\*](#)
- [\*HAL\\_UART\\_GetError\(\)\*](#)

#### 44.2.6 HAL\_UART\_Init

Function Name	<b>HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)</b>
Function Description	Initializes the UART 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>HAL status</li> </ul>

#### 44.2.7 HAL\_HalfDuplex\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)</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>HAL status</li> </ul>

#### 44.2.8 HAL\_LIN\_Init

Function Name	<b>HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)</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:            UART_LINBREAKDETECTLENGTH_10B: 10-bit break detection            UART_LINBREAKDETECTLENGTH_11B: 11-bit break detection         </li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 44.2.9 HAL\_MultiProcessor\_Init

Function Name	<b>HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)</b>
Function Description	Initializes the Multi-Processor 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>Address:</b> USART address</li> </ul>

- **WakeUpMethod:** specifies the USART wake-up method.  
This parameter can be one of the following values:  
UART\_WAKEUPMETHOD\_IDLELINE: Wake-up by an idle line detection  
UART\_WAKEUPMETHOD\_ADDRESSMARK: Wake-up by an address mark

Return values • HAL status

#### 44.2.10 HAL\_UART\_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)</code>
Function Description	DeInitializes the USART peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</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>• HAL status</li> </ul>

#### 44.2.11 HAL\_UART\_MspInit

Function Name	<code>void HAL_UART_MspInit (UART_HandleTypeDef * huart)</code>
Function Description	USART MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</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>

#### 44.2.12 HAL\_UART\_MspDeInit

Function Name	<code>void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)</code>
Function Description	USART MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</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>

#### 44.2.13 HAL\_UART\_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Sends an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART 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>• HAL status</li> </ul>
---------------	--

#### 44.2.14 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 44.2.15 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 44.2.16 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> <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>• HAL status</li> </ul>

#### 44.2.17 HAL\_UART\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_DMA</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> </ul>

- **pData:** Pointer to data buffer
  - **Size:** Amount of data to be sent
- Return values
- HAL status

#### 44.2.18 HAL\_UART\_Receive\_DMA

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_UART_Receive_DMA<br/>(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> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be received</li> </ul> |
| Return values        | • HAL status   |
| Notes                | <ul style="list-style-type: none"> <li>• When the UART parity is enabled (PCE = 1) the data received contain the parity bit.</li> </ul>  |

#### 44.2.19 HAL\_UART\_DMAPause

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_UART_DMAPause<br/>(UART_HandleTypeDef * huart)</b>  |
| Function Description | Pauses the DMA Transfer.   |
| 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> |
| Return values        | • HAL status   |

#### 44.2.20 HAL\_UART\_DMAResume

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_UART_DMAResume<br/>(UART_HandleTypeDef * huart)</b>   |
| Function Description | Resumes the DMA Transfer.  |
| 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> |
| Return values        | • HAL status   |

#### 44.2.21 HAL\_UART\_DMAStop

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_UART_DMAStop<br/>(UART_HandleTypeDef * huart)</b>   |
| Function Description | Stops the DMA Transfer.  |
| 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> |

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

#### 44.2.22 HAL\_UART\_IRQHandler

Function Name	<b>void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)</b>
Function Description	This function handles UART interrupt request.
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>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 44.2.23 HAL\_UART\_TxCpltCallback

Function Name	<b>void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)</b>
Function Description	Tx Transfer completed callbacks.
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>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 44.2.24 HAL\_UART\_TxHalfCpltCallback

Function Name	<b>void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)</b>
Function Description	Tx Half Transfer completed callbacks.
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>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 44.2.25 HAL\_UART\_RxCpltCallback

Function Name	<b>void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)</b>
Function Description	Rx Transfer completed callbacks.
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>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 44.2.26 HAL\_UART\_RxHalfCpltCallback

Function Name	<b>void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> pointer to a UART_HandleTypeDef structure that</li> </ul>

contains the configuration information for the specified UART module.

Return values

- None

#### 44.2.27 HAL\_UART\_ErrorCallback

Function Name      **void HAL\_UART\_ErrorCallback (UART\_HandleTypeDef \* huart)**

Function Description      [UART error callbacks.](#)

Parameters

- **huart:** pointer to a `UART_HandleTypeDef` structure that contains the configuration information for the specified UART module.

Return values

- None

#### 44.2.28 HAL\_LIN\_SendBreak

Function Name      **HAL\_StatusTypeDef HAL\_LIN\_SendBreak (UART\_HandleTypeDef \* huart)**

Function Description      [Transmits break characters.](#)

Parameters

- **huart:** pointer to a `UART_HandleTypeDef` structure that contains the configuration information for the specified UART module.

Return values

- HAL status

#### 44.2.29 HAL\_MultiProcessor\_EnterMuteMode

Function Name      **HAL\_StatusTypeDef HAL\_MultiProcessor\_EnterMuteMode (UART\_HandleTypeDef \* huart)**

Function Description      [Enters the UART in mute mode.](#)

Parameters

- **huart:** pointer to a `UART_HandleTypeDef` structure that contains the configuration information for the specified UART module.

Return values

- HAL status

#### 44.2.30 HAL\_MultiProcessor\_ExitMuteMode

Function Name      **HAL\_StatusTypeDef HAL\_MultiProcessor\_ExitMuteMode (UART\_HandleTypeDef \* huart)**

Function Description      [Exits the UART mute mode: wake up software.](#)

Parameters

- **huart:** pointer to a `UART_HandleTypeDef` structure that contains the configuration information for the specified UART module.

Return values

- HAL status

#### 44.2.31 HAL\_HalfDuplex\_EnableTransmitter

Function Name      **HAL\_StatusTypeDef HAL\_HalfDuplex\_EnableTransmitter**

**(UART\_HandleTypeDef \* huart)**

Function Description	Enables the UART transmitter and disables the UART receiver.
Parameters	<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>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**44.2.32 HAL\_HalfDuplex\_EnableReceiver**

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver</b> <b>(UART_HandleTypeDef * 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 UART_HandleTypeDef structure that contains the configuration information for the specified UART module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**44.2.33 HAL\_UART\_GetState**

Function Name	<b>HAL_UART_StateTypeDef HAL_UART_GetState</b> <b>(UART_HandleTypeDef * huart)</b>
Function Description	Returns the UART state.
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>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

**44.2.34 HAL\_UART\_GetError**

Function Name	<b>uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)</b>
Function Description	Return the UART error code.
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.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• UART Error Code</li> </ul>

**44.3 UART Firmware driver defines****44.3.1 UART*****UART Error Code***

HAL_UART_ERROR_NONE	No error
HAL_UART_ERROR_PE	Parity error
HAL_UART_ERROR_NE	Noise error
HAL_UART_ERROR_FE	Frame error
HAL_UART_ERROR_ORE	Overrun error

`HAL_UART_ERROR_DMA` DMA transfer error

***UART Exported Macros***

`_HAL_UART_RESET_HANDLE_STATE`

**Description:**

- Reset UART handle state.

**Parameters:**

- `_HANDLE_`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`_HAL_UART_FLUSH_DRREGISTER`

**Description:**

- Flushes the UART DR register.

**Parameters:**

- `_HANDLE_`: specifies the UART Handle.

`_HAL_UART_GET_FLAG`

**Description:**

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

**Parameters:**

- `_HANDLE_`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - `UART_FLAG_CTS`: CTS Change flag (not available for `UART4` and `UART5`)
  - `UART_FLAG_LBD`: LIN Break detection flag
  - `UART_FLAG_TXE`: Transmit data register empty flag
  - `UART_FLAG_TC`: Transmission Complete flag
  - `UART_FLAG_RXNE`: Receive data register not empty flag
  - `UART_FLAG_IDLE`: Idle Line detection flag
  - `UART_FLAG_ORE`: Overrun Error flag
  - `UART_FLAG_NE`: Noise Error flag
  - `UART_FLAG_FE`: Framing

- Error flag
- UART\_FLAG\_PE: Parity Error flag

**Return value:**

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

[\\_\\_HAL\\_UART\\_CLEAR\\_FLAG](#)**Description:**

- Clears the specified UART pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- `__FLAG__`: specifies the flag to check. This parameter can be any combination of the following values:
  - `UART_FLAG_CTS`: CTS Change flag (not available for `UART4` and `UART5`).
  - `UART_FLAG_LBD`: LIN Break detection flag.
  - `UART_FLAG_TC`: Transmission Complete flag.
  - `UART_FLAG_RXNE`: Receive data register not empty flag.

**Return value:**

- None

**Notes:**

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (Overrun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to `USART_SR` register followed by a read operation to `USART_DR` register. RXNE flag can be also cleared by a read to the `USART_DR` register. TC flag can be also cleared by software sequence: a read operation to `USART_SR` register followed by a write operation to `USART_DR` register. TXE flag is cleared only by a write to the `USART_DR` register.

[\\_\\_HAL\\_UART\\_CLEAR\\_PEFLAG](#)**Description:**

- Clear the UART PE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_UART_CLEAR_FEFLAG`

**Description:**

- Clear the UART FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_UART_CLEAR_NEFLAG`

**Description:**

- Clear the UART NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_UART_CLEAR_OREFLAG`

**Description:**

- Clear the UART ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`__HAL_UART_CLEAR_IDLEFLAG`

**Description:**

- Clear the UART IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART

Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.

**Return value:**

- None

`UART_IT_MASK`

**Description:**

- Enable the specified UART interrupt.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- `__INTERRUPT__`: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - `UART_IT_CTS`: CTS change interrupt
  - `UART_IT_LBD`: LIN Break detection interrupt
  - `UART_IT_TXE`: Transmit Data Register empty interrupt
  - `UART_IT_TC`: Transmission complete interrupt
  - `UART_IT_RXNE`: Receive Data register not empty interrupt
  - `UART_IT_IDLE`: Idle line detection interrupt
  - `UART_IT_PE`: Parity Error interrupt
  - `UART_IT_ERR`: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

`__HAL_UART_ENABLE_IT`

`__HAL_UART_DISABLE_IT`

**Description:**

- Disable the specified UART interrupt.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle. This parameter can be `UARTx` where `x`: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- `__INTERRUPT__`: specifies the

UART interrupt source to disable.

This parameter can be one of the following values:

- UART\_IT\_CTS: CTS change interrupt
- UART\_IT\_LBD: LIN Break detection interrupt
- UART\_IT\_TXE: Transmit Data Register empty interrupt
- UART\_IT\_TC: Transmission complete interrupt
- UART\_IT\_RXNE: Receive Data register not empty interrupt
- UART\_IT\_IDLE: Idle line detection interrupt
- UART\_IT\_PE: Parity Error interrupt
- UART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

[\\_\\_HAL\\_UART\\_GET\\_IT\\_SOURCE](#)

- Checks whether the specified UART interrupt has occurred or not.

**Parameters:**

- [\\_\\_HANDLE\\_\\_](#): specifies the UART Handle. This parameter can be UARTx where x: 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or UART peripheral.
- [\\_\\_IT\\_\\_](#): specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART\_IT\_CTS: CTS change interrupt (not available for UART4 and UART5)
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty interrupt
  - UART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_ERR: Error interrupt

**Return value:**

- The: new state of [\\_\\_IT\\_\\_](#) (TRUE or

FALSE).

#### `_HAL_UART_HWCONTROL_CTS_ENABLE`

##### **Description:**

- Enable CTS flow control This macro allows to enable CTS hardware flow control for a given UART instance, without need to call

##### **Parameters:**

- `_HANDLE_`: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

##### **Return value:**

- None

##### **Notes:**

- As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of `HAL_UART_Init()`) macro could only be called when corresponding UART instance is disabled (i.e `_HAL_UART_DISABLE(_HANDLE_)`) and should be followed by an Enable macro (i.e `_HAL_UART_ENABLE(_HANDLE_)`).

#### `_HAL_UART_HWCONTROL_CTS_DISABLE`

##### **Description:**

- Disable CTS flow control This macro allows to disable CTS hardware flow control for a given UART instance, without need to call

##### **Parameters:**

- `_HANDLE_`: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

##### **Return value:**

- None

##### **Notes:**

- As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call

of HAL\_UART\_Init() )macro could only be called when corresponding UART instance is disabled (i.e \_\_HAL\_UART\_DISABLE(\_\_HANDLE)) and should be followed by an Enable macro (i.e \_\_HAL\_UART\_ENABLE(\_\_HANDLE)).

#### `__HAL_UART_HWCONTROL_RTS_ENABLE`

##### **Description:**

- Enable RTS flow control This macro allows to enable RTS hardware flow control for a given UART instance, without need to call

##### **Parameters:**

- `__HANDLE__`: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

##### **Return value:**

- None

##### **Notes:**

- As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL\_UART\_Init() )macro could only be called when corresponding UART instance is disabled (i.e \_\_HAL\_UART\_DISABLE(\_\_HANDLE)) and should be followed by an Enable macro (i.e \_\_HAL\_UART\_ENABLE(\_\_HANDLE)).

#### `__HAL_UART_HWCONTROL_RTS_DISABLE`

##### **Description:**

- Disable RTS flow control This macro allows to disable RTS hardware flow control for a given UART instance, without need to call

##### **Parameters:**

- `__HANDLE__`: specifies the UART Handle. The Handle Instance can be USART1, USART2 or LPUART.

##### **Return value:**

- None

##### **Notes:**

- As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : USART instance should have already been initialised (through call of HAL\_UART\_Init() )macro could only be called when corresponding USART instance is disabled (i.e \_\_HAL\_UART\_DISABLE(\_\_HANDLE)) and should be followed by an Enable macro (i.e \_\_HAL\_UART\_ENABLE(\_\_HANDLE)).

`__HAL_UART_ONE_BIT_SAMPLE_ENABLE`

**Description:**

- macros to enables the UART's one bit sample method

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_ONE_BIT_SAMPLE_DISABLE`

**Description:**

- macros to disables the UART's one bit sample method

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_ENABLE`

**Description:**

- Enable UART.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_DISABLE`

**Description:**

- Disable UART.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

***UART Flags***

UART\_FLAG\_CTS  
UART\_FLAG\_LBD  
UART\_FLAG\_TXE  
UART\_FLAG\_TC  
UART\_FLAG\_RXNE  
UART\_FLAG\_IDLE  
UART\_FLAG\_ORE  
UART\_FLAG\_NE  
UART\_FLAG\_FE  
UART\_FLAG\_PE

***UART Hardware Flow Control***

UART\_HWCONTROL\_NONE  
UART\_HWCONTROL\_RTS  
UART\_HWCONTROL\_CTS  
UART\_HWCONTROL\_RTS\_CTS

***UART Interrupt Definitions***

UART\_IT\_PE  
UART\_IT\_TXE  
UART\_IT\_TC  
UART\_IT\_RXNE  
UART\_IT\_IDLE  
UART\_IT\_LBD  
UART\_IT\_CTS  
UART\_IT\_ERR

***UART LIN Break Detection Length***

UART\_LINBREAKDETECTLENGTH\_10B  
UART\_LINBREAKDETECTLENGTH\_11B

***UART Transfer Mode***

UART\_MODE\_RX  
UART\_MODE\_TX  
UART\_MODE\_TX\_RX

***UART Over Sampling***

UART\_OVERSAMPLING\_16

UART\_OVERSAMPLING\_8  
**UART Parity**  
UART\_PARITY\_NONE  
UART\_PARITY EVEN  
UART\_PARITY ODD  
**UART Private Constants**  
UART\_TIMEOUT\_VALUE  
UART\_CR1\_REG\_INDEX  
UART\_CR2\_REG\_INDEX  
UART\_CR3\_REG\_INDEX  
**UART Private Macros**  
IS\_UART\_WORD\_LENGTH  
IS\_UART\_LIN\_WORD\_LENGTH  
IS\_UART\_STOPBITS  
IS\_UART\_PARITY  
IS\_UART\_HARDWARE\_FLOW\_CONTROL  
IS\_UART\_MODE  
IS\_UART\_STATE  
IS\_UART\_OVERSAMPLING  
IS\_UART\_LIN\_OVERSAMPLING  
IS\_UART\_LIN\_BREAK\_DETECT\_LENGTH  
IS\_UART\_WAKEUPMETHOD  
IS\_UART\_BAUDRATE  
IS\_UART\_ADDRESS  
UART\_DIV\_SAMPLING16  
UART\_DIVMANT\_SAMPLING16  
UART\_DIVFRAQ\_SAMPLING16  
UART\_BRR\_SAMPLING16  
UART\_DIV\_SAMPLING8  
UART\_DIVMANT\_SAMPLING8  
UART\_DIVFRAQ\_SAMPLING8  
UART\_BRR\_SAMPLING8  
**UART State**  
UART\_STATE\_DISABLE  
UART\_STATE\_ENABLE  
**UART Number of Stop Bits**

UART\_STOPBITS\_1

UART\_STOPBITS\_2

***UART Wakeup Functions***

UART\_WAKEUPMETHOD\_IDLELINE

UART\_WAKEUPMETHOD\_ADDRESSMARK

***UART Word Length***

UART\_WORDLENGTH\_8B

UART\_WORDLENGTH\_9B

## 45 HAL USART Generic Driver

### 45.1 USART Firmware driver registers structures

#### 45.1.1 USART\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t CLKLastBit*

##### Field Documentation

- ***uint32\_t USART\_InitTypeDef::BaudRate***  
This member configures the Usart communication baud rate. The baud rate is computed using the following formula:  

$$\text{IntegerDivider} = ((\text{PCLKx}) / (8 * (\text{husart->Init.BaudRate})))$$

$$\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{uint32_t}) \text{ IntegerDivider})) * 8) + 0.5$$
- ***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\*\*](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32\_t USART\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [\*\*USART\\_Mode\*\*](#)
- ***uint32\_t USART\_InitTypeDef::CLKPolarity***  
Specifies the steady state of the serial clock. This parameter can be a value of [\*\*USART\\_Clock\\_Polarity\*\*](#)
- ***uint32\_t USART\_InitTypeDef::CLKPhase***  
Specifies the clock transition on which the bit capture is made. This parameter can be a value of [\*\*USART\\_Clock\\_Phase\*\*](#)
- ***uint32\_t USART\_InitTypeDef::CLKLastBit***  
Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of [\*\*USART\\_Last\\_Bit\*\*](#)

## 45.1.2 USART\_HandleTypeDef

### Data Fields

- *USART\_TypeDef \* Instance*
- *USART\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *uint16\_t TxXferSize*
- *\_IO uint16\_t TxXferCount*
- *uint8\_t \* pRxBuffPtr*
- *uint16\_t RxXferSize*
- *\_IO uint16\_t RxXferCount*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *HAL\_LockTypeDef Lock*
- *\_IO HAL\_USART\_StateTypeDef State*
- *\_IO uint32\_t ErrorCode*

### Field Documentation

- *USART\_TypeDef\* USART\_HandleTypeDef::Instance*
- *USART\_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 uint32\_t USART\_HandleTypeDef::ErrorCode*

## 45.2 USART Firmware driver API description

### 45.2.1 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 DMA 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 usart 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 customized HAL\_USART\_MspInit(&husart) API. The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros \_\_HAL\_USART\_ENABLE\_IT() and \_\_HAL\_USART\_DISABLE\_IT() inside the transmit and receive process.
- 5. Three operation modes are available within this driver :

### **Polling mode IO operation**

- Send an amount of data in blocking mode using HAL\_USART\_Transmit()
- Receive an amount of data in blocking mode using HAL\_USART\_Receive()

### **Interrupt mode IO operation**

- Send an amount of data in non blocking mode using HAL\_USART\_Transmit\_IT()
- At transmission end of transfer HAL\_USART\_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_USART\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_USART\_Receive\_IT()
- At reception end of transfer HAL\_USART\_RxHalfCpltCallback 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

#### 45.2.2

### Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), please refer to Reference manual for possible USART frame formats.
  - 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)).

This section contains the following APIs:

- [\*\*HAL\\_USART\\_Init\(\)\*\*](#)
- [\*\*HAL\\_USART\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_MspDeInit\(\)\*\*](#)

### 45.2.3 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\_TxRx\_CpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process. The HAL\_USART\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
  - HAL\_USART\_Transmit() in simplex mode
  - HAL\_USART\_Receive() in full duplex receive only
  - HAL\_USART\_TransmitReceive() in full duplex mode
3. Non Blocking mode APIs with Interrupt are :
  - HAL\_USART\_Transmit\_IT() in simplex mode
  - HAL\_USART\_Receive\_IT() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_IT() in full duplex mode
  - HAL\_USART\_IRQHandler()
4. Non Blocking mode functions with DMA are :
  - HAL\_USART\_Transmit\_DMA() in simplex mode
  - HAL\_USART\_Receive\_DMA() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_DMA() in full duplex mode
  - HAL\_USART\_DMAPause()
  - HAL\_USART\_DMAResume()
  - HAL\_USART\_DMAMode()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_USART\_TxHalfCpltCallback()
  - HAL\_USART\_TxCpltCallback()
  - HAL\_USART\_RxHalfCpltCallback()
  - HAL\_USART\_RxCpltCallback()
  - HAL\_USART\_ErrorCallback()
  - HAL\_USART\_TxRx\_CpltCallback()

This section contains the following APIs:

- [\*\*HAL\\_USART\\_Transmit\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Receive\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TransmitReceive\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Transmit\\_IT\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Receive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TransmitReceive\\_IT\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Transmit\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_USART\\_Receive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_USART\\_TransmitReceive\\_DMA\(\)\*\*](#)
- [\*\*HAL\\_USART\\_DMAPause\(\)\*\*](#)

- [`HAL\_USART\_DMAResume\(\)`](#)
- [`HAL\_USART\_DMAStop\(\)`](#)
- [`HAL\_USART\_IRQHandler\(\)`](#)
- [`HAL\_USART\_TxCpltCallback\(\)`](#)
- [`HAL\_USART\_TxHalfCpltCallback\(\)`](#)
- [`HAL\_USART\_RxCpltCallback\(\)`](#)
- [`HAL\_USART\_RxHalfCpltCallback\(\)`](#)
- [`HAL\_USART\_TxRxCpltCallback\(\)`](#)
- [`HAL\_USART\_ErrorCallback\(\)`](#)

#### 45.2.4 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.

This section contains the following APIs:

- [`HAL\_USART\_GetState\(\)`](#)
- [`HAL\_USART\_GetError\(\)`](#)

#### 45.2.5 HAL\_USART\_Init

Function Name	<code>HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * huart)</code>
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>• HAL status</li> </ul>

#### 45.2.6 HAL\_USART\_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_USART_DelInit (USART_HandleTypeDef * huart)</code>
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>• HAL status</li> </ul>

#### 45.2.7 HAL\_USART\_MspInit

Function Name	<code>void HAL_USART_MspInit (USART_HandleTypeDef * huart)</code>
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</li> </ul>

USART module.

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

#### 45.2.8 HAL\_USART\_MspDeInit

Function Name	<code>void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)</code>
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>

#### 45.2.9 HAL\_USART\_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)</code>
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> <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>HAL status</li> </ul>

#### 45.2.10 HAL\_USART\_Receive

Function Name	<code>HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</code>
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>HAL status</li> </ul>

#### 45.2.11 HAL\_USART\_TransmitReceive

Function Name	<code>HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)</code>
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>HAL status</li> </ul>

#### 45.2.12 HAL\_USART\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit_IT(USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t 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>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>The USART errors are not managed to avoid the overrun error.</li> </ul>

#### 45.2.13 HAL\_USART\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive_IT(USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t 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>HAL status</li> </ul>

#### 45.2.14 HAL\_USART\_TransmitReceive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive_IT(USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)</b>
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> </ul>

- **Size:** Amount of data to be received
- HAL status

#### 45.2.15 HAL\_USART\_Transmit\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef *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>• HAL status</li> </ul>

#### 45.2.16 HAL\_USART\_Receive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef *husart, uint8_t *pRxData, uint16_t Size)</code>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
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>

#### 45.2.17 HAL\_USART\_TransmitReceive\_DMA

Function Name	<code>HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef *husart, uint8_t *pTxData, uint8_t * pRxData, uint16_t Size)</code>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

- Notes**
- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

#### 45.2.18 HAL\_USART\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_USART_DM_PAUSE (USART_HandleTypeDef * <b>husart</b>)</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>HAL status</li> </ul>

#### 45.2.19 HAL\_USART\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * <b>husart</b>)</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>HAL status</li> </ul>

#### 45.2.20 HAL\_USART\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * <b>husart</b>)</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>HAL status</li> </ul>

#### 45.2.21 HAL\_USART\_IRQHandler

Function Name	<b>void HAL_USART_IRQHandler (USART_HandleTypeDef * <b>husart</b>)</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>

#### 45.2.22 HAL\_USART\_TxCpltCallback

Function Name	<b>void HAL_USART_TxCpltCallback (USART_HandleTypeDef * <b>husart</b>)</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>

#### 45.2.23 HAL\_USART\_TxHalfCpltCallback

Function Name	<b>void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)</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>

#### 45.2.24 HAL\_USART\_RxCpltCallback

Function Name	<b>void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)</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>

#### 45.2.25 HAL\_USART\_RxHalfCpltCallback

Function Name	<b>void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)</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>

#### 45.2.26 HAL\_USART\_TxRxCpltCallback

Function Name	<b>void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)</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>

#### 45.2.27 HAL\_USART\_ErrorCallback



Function Name	<b>void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)</b>
Function Description	USART error 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>

#### 45.2.28 HAL\_USART\_GetState

Function Name	<b>HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)</b>
Function Description	Returns the USART state.
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>• HAL state</li> </ul>

#### 45.2.29 HAL\_USART\_GetError

Function Name	<b>uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)</b>
Function Description	Return the USART error code.
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.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• USART Error Code</li> </ul>

### 45.3 USART Firmware driver defines

#### 45.3.1 USART

##### ***USART Clock***

USART\_CLOCK\_DISABLE

USART\_CLOCK\_ENABLE

##### ***USART Clock Phase***

USART\_PHASE\_1EDGE

USART\_PHASE\_2EDGE

##### ***USART Clock Polarity***

USART\_POLARITY\_LOW

USART\_POLARITY\_HIGH

##### ***USART Error Code***

HAL\_USART\_ERROR\_NONE No error

HAL_USART_ERROR_PE	Parity error
HAL_USART_ERROR_NE	Noise error
HAL_USART_ERROR_FE	Frame error
HAL_USART_ERROR_ORE	Overrun error
HAL_USART_ERROR_DMA	DMA transfer error

**USART Exported Macros**`_HAL_USART_RESET_HANDLE_STATE`**Description:**

- Reset USART handle state.

**Parameters:**

- `_HANDLE_`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.

**Return value:**

- None

`_HAL_USART_GET_FLAG`**Description:**

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

**Parameters:**

- `_HANDLE_`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.
- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
  - USART\_FLAG\_TXE: Transmit data register empty flag
  - USART\_FLAG\_TC: Transmission Complete flag
  - USART\_FLAG\_RXNE: Receive data register not empty flag
  - USART\_FLAG\_IDLE: Idle Line detection flag
  - USART\_FLAG\_ORE: Overrun Error flag
  - USART\_FLAG\_NE: Noise Error flag
  - USART\_FLAG\_FE: Framing Error flag
  - USART\_FLAG\_PE: Parity Error flag

**Return value:**

- The new state of `_FLAG_`

(TRUE or FALSE).

### \_HAL\_USART\_CLEAR\_FLAG

#### **Description:**

- Clears the specified Smartcard pending flags.

#### **Parameters:**

- \_HANDLE\_: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.
- \_FLAG\_: specifies the flag to check. This parameter can be any combination of the following values:
  - USART\_FLAG\_TC: Transmission Complete flag.
  - USART\_FLAG\_RXNE: Receive data register not empty flag.

#### **Return value:**

- None

#### **Notes:**

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (Overrun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART\_SR register followed by a read operation to USART\_DR register. RXNE flag can be also cleared by a read to the USART\_DR register. TC flag can be also cleared by software sequence: a read operation to USART\_SR register followed by a write operation to USART\_DR register. TXE flag is cleared only by a write to the USART\_DR register.

### \_HAL\_USART\_CLEAR\_PEFLAG

#### **Description:**

- Clear the USART PE pending flag.

#### **Parameters:**

- \_HANDLE\_: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.

#### **Return value:**

- None

### \_HAL\_USART\_CLEAR\_FEFLAG

#### **Description:**

- Clear the USART FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.

**Return value:**

- None

`__HAL_USART_CLEAR_NEFLAG`

- Clear the USART NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.

**Return value:**

- None

`__HAL_USART_CLEAR_OREFLAG`

- Clear the USART ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.

**Return value:**

- None

`__HAL_USART_CLEAR_IDLEFLAG`

- Clear the USART IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.

**Return value:**

- None

`__HAL_USART_ENABLE_IT`

- Enables or disables the specified USART interrupts.

**Parameters:**

- `__HANDLE__`: specifies the

USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.

- \_\_INTERRUPT\_\_: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_PE: Parity Error interrupt
  - USART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error) This parameter can be: ENABLE or DISABLE.

#### **Return value:**

- None

\_HAL\_USART\_DISABLE\_IT  
\_HAL\_USART\_GET\_IT\_SOURCE

#### **Description:**

- Checks whether the specified USART interrupt has occurred or not.

#### **Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle. This parameter can be USARTx where x: 1, 2, 3 or 6 to select the USART peripheral.
- \_\_IT\_\_: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_ERR: Error interrupt
  - USART\_IT\_PE: Parity Error

---

interrupt

**Return value:**

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

`__HAL_USART_ONE_BIT_SAMPLE_ENABLE`**Description:**

- Macro to enable the USART's one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle.

**Return value:**

- None

`__HAL_USART_ONE_BIT_SAMPLE_DISABLE`**Description:**

- Macro to disable the USART's one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle.

**Return value:**

- None

`__HAL_USART_ENABLE`**Description:**

- Enable USART.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

`__HAL_USART_DISABLE`**Description:**

- Disable USART.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

**Return value:**

- None

***USART Flags***

USART\_FLAG\_TXE  
USART\_FLAG\_TC  
USART\_FLAG\_RXNE  
USART\_FLAG\_IDLE  
USART\_FLAG\_ORE  
USART\_FLAG\_NE  
USART\_FLAG\_FE  
USART\_FLAG\_PE

***USART Interrupts Definition***

USART\_IT\_PE  
USART\_IT\_TXE  
USART\_IT\_TC  
USART\_IT\_RXNE  
USART\_IT\_IDLE  
USART\_IT\_LBD  
USART\_IT\_CTS  
USART\_IT\_ERR

***USART Last Bit***

USART\_LASTBIT\_DISABLE  
USART\_LASTBIT\_ENABLE

***USART Mode***

USART\_MODE\_RX  
USART\_MODE\_TX  
USART\_MODE\_TX\_RX

***USART NACK State***

USART\_NACK\_ENABLE  
USART\_NACK\_DISABLE

***USART Parity***

USART\_PARITY\_NONE  
USART\_PARITY\_EVEN  
USART\_PARITY\_ODD

***USART Private Constants***

DUMMY\_DATA  
USART\_TIMEOUT\_VALUE  
USART\_IT\_MASK  
USART\_CR1\_REG\_INDEX

USART\_CR2\_REG\_INDEX

USART\_CR3\_REG\_INDEX

***USART Private Macros***

IS\_USART\_NACK\_STATE

IS\_USART\_LASTBIT

IS\_USART\_PHASE

IS\_USART\_POLARITY

IS\_USART\_CLOCK

IS\_USART\_WORD\_LENGTH

IS\_USART\_STOPBITS

IS\_USART\_PARITY

IS\_USART\_MODE

IS\_USART\_BAUDRATE

USART\_DIV

USART\_DIVMANT

USART\_DIVFRAQ

USART\_BRR

***USART Number of Stop Bits***

USART\_STOPBITS\_1

USART\_STOPBITS\_0\_5

USART\_STOPBITS\_2

USART\_STOPBITS\_1\_5

***USART Word Length***

USART\_WORDLENGTH\_8B

USART\_WORDLENGTH\_9B

## 46 HAL WWDG Generic Driver

### 46.1 WWDG Firmware driver registers structures

#### 46.1.1 WWDG\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Window*
- *uint32\_t Counter*

##### Field Documentation

- ***uint32\_t WWDG\_InitTypeDef::Prescaler***  
Specifies the prescaler value of the WWDG. This parameter can be a value of [WWDG\\_Prescaler](#)
- ***uint32\_t WWDG\_InitTypeDef::Window***  
Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number lower than Max\_Data = 0x80
- ***uint32\_t WWDG\_InitTypeDef::Counter***  
Specifies the WWDG free-running downcounter value. This parameter must be a number between Min\_Data = 0x40 and Max\_Data = 0x7F

#### 46.1.2 WWDG\_HandleTypeDefDef

##### 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

## 46.2 WWDG Firmware driver API description

### 46.2.1 WWDG specific features

Once enabled the WWdg generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWdg cannot be disabled except by a system reset.
- WWDRST flag in RCC\_CSR register can be used to inform when a WWdg reset occurs.
- The WWdg counter input clock is derived from the APB clock divided by a programmable prescaler.
- WWdg clock (Hz) = PCLK1 / (4096 \* Prescaler)
- WWdg timeout (mS) = 1000 \* Counter / WWdg clock
- WWdg Counter refresh is allowed between the following limits :
  - min time (mS) = 1000 \* (Counter \_ Window) / WWdg clock
  - max time (mS) = 1000 \* (Counter \_ 0x40) / WWdg clock
- Min-max timeout value at 50 MHz(PCLK1): 81.9 us / 41.9 ms

### 46.2.2 How to use this driver

- Enable WWdg APB1 clock using \_\_HAL\_RCC\_WWDG\_CLK\_ENABLE().
- Set the WWdg prescaler, refresh window and counter value using HAL\_WWDG\_Init() function.
- Start the WWdg using HAL\_WWDG\_Start() function. When the WWdg is enabled the counter value should be configured to a value greater than 0x40 to prevent generating an immediate reset.
- Optionally you can enable the Early Wakeup Interrupt (EWI) which is generated when the counter reaches 0x40, and then start the WWdg using HAL\_WWDG\_Start\_IT(). At EWI HAL\_WWDG\_WakeupCallback is executed and user can add his own code by customization of function pointer HAL\_WWDG\_WakeupCallback Once enabled, EWI interrupt cannot be disabled except by a system reset.
- Then the application program must refresh the WWdg counter at regular intervals during normal operation to prevent an MCU reset, using HAL\_WWDG\_Refresh() function. This operation must occur only when the counter is lower than the refresh window value already programmed.

#### WWdg HAL driver macros list

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

- \_\_HAL\_WWDG\_ENABLE: Enable the WWdg peripheral
- \_\_HAL\_WWDG\_GET\_FLAG: Get the selected WWdg's flag status
- \_\_HAL\_WWDG\_CLEAR\_FLAG: Clear the WWdg's pending flags
- \_\_HAL\_WWDG\_ENABLE\_IT: Enables the WWdg early wake-up interrupt

### 46.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the WWDG according to the specified parameters in the WWDG\_InitTypeDef and create the associated handle
- DeInitialize the WWDG peripheral
- Initialize the WWDG MSP
- DeInitialize the WWDG MSP

This section contains the following APIs:

- [\*HAL\\_WWDG\\_Init\(\)\*](#)
- [\*HAL\\_WWDG\\_DeInit\(\)\*](#)
- [\*HAL\\_WWDG\\_MspInit\(\)\*](#)
- [\*HAL\\_WWDG\\_MspDeInit\(\)\*](#)
- [\*HAL\\_WWDG\\_WakeupCallback\(\)\*](#)

#### 46.2.4 IO operation functions

This section provides functions allowing to:

- Start the WWDG.
- Refresh the WWDG.
- Handle WWDG interrupt request.

This section contains the following APIs:

- [\*HAL\\_WWDG\\_Start\(\)\*](#)
- [\*HAL\\_WWDG\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_WWDG\\_Refresh\(\)\*](#)
- [\*HAL\\_WWDG\\_IRQHandler\(\)\*](#)
- [\*HAL\\_WWDG\\_WakeupCallback\(\)\*](#)

#### 46.2.5 Peripheral State functions

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

This section contains the following APIs:

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

#### 46.2.6 HAL\_WWDG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwdg)</b>
Function Description	Initializes the WWDG according to the specified parameters in the WWDG_InitTypeDef and creates the associated handle.
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>• HAL status</li> </ul>

#### 46.2.7 HAL\_WWDG\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_DeInit (WWDG_HandleTypeDef * hwdg)</b>
Function Description	DeInitializes the WWDG peripheral.
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</li> </ul>

WWDG module.

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

#### 46.2.8 HAL\_WWDG\_MspInit

Function Name	<b>void HAL_WWDG_MspInit (WWDG_HandleTypeDefDef * hwdg)</b>
Function Description	Initializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg:</b> pointer to a WWDG_HandleTypeDefDef 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>

#### 46.2.9 HAL\_WWDG\_MspDeInit

Function Name	<b>void HAL_WWDG_MspDeInit (WWDG_HandleTypeDefDef * hwdg)</b>
Function Description	Deinitializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg:</b> pointer to a WWDG_HandleTypeDefDef 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>

#### 46.2.10 HAL\_WWDG\_WakeupCallback

Function Name	<b>void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDefDef * hwdg)</b>
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg:</b> pointer to a WWDG_HandleTypeDefDef 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>

#### 46.2.11 HAL\_WWDG\_Start

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Start (WWDG_HandleTypeDefDef * hwdg)</b>
Function Description	Starts the WWDG.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwdg:</b> pointer to a WWDG_HandleTypeDefDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 46.2.12 HAL\_WWDG\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDefDef * hwdg)</b>
Function Description	Starts the WWDG with interrupt enabled.

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>HAL status</li> </ul>

#### 46.2.13 HAL\_WWDG\_Refresh

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwdwg, uint32_t Counter)</b>
Function Description	Refreshes the WWDG.
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> <li><b>Counter:</b> value of counter to put in WWDG counter</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 46.2.14 HAL\_WWDG\_IRQHandler

Function Name	<b>void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwdwg)</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 __HAL_WWDG_ENABLE_IT() macro. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.</li> </ul>

#### 46.2.15 HAL\_WWDG\_WakeupCallback

Function Name	<b>void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwdwg)</b>
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>

#### 46.2.16 HAL\_WWDG\_GetState

Function Name	<b>HAL_WWDG_StateTypeDef HAL_WWDG_GetState (WWDG_HandleTypeDef * hwdwg)</b>
---------------	---

Function Description	Returns the WWDG state.
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>• HAL state</li> </ul>

## 46.3 WWDG Firmware driver defines

### 46.3.1 WWDG

**WWDG BitAddress**

**WWDG\_CFR\_BASE**

**WWDG Exported Macros**

`_HAL_WWDG_RESET_HANDLE_STATE` **Description:**

- Reset WWDG handle state.

**Parameters:**

- `_HANDLE_`: WWDG handle

**Return value:**

- None

**Description:**

- Enables the WWDG peripheral.

**Parameters:**

- `_HANDLE_`: WWDG handle

**Return value:**

- None

**Description:**

- Disables the WWDG peripheral.

**Parameters:**

- `_HANDLE_`: WWDG handle

**Return value:**

- None

**Notes:**

- WARNING: This is a dummy macro for HAL code alignment. Once enable, WWDG Peripheral cannot be disabled except by a system reset.

**Description:**

- Gets the selected WWDG's it status.

**Parameters:**

- `_HANDLE_`: WWDG handle

- \_\_INTERRUPT\_\_: specifies the interrupt to check. This parameter can be one of the following values:
  - WWDG\_FLAG\_EWIF: Early wakeup interrupt IT

**Return value:**

- The new state of WWDG\_FLAG (SET or RESET).

[\\_\\_HAL\\_WWDG\\_CLEAR\\_IT](#)**Description:**

- Clear the WWDG's interrupt pending bits to clear the selected interrupt pending bits.

**Parameters:**

- \_\_HANDLE\_\_: WWDG handle
- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG\_FLAG\_EWIF: Early wakeup interrupt flag

[\\_\\_HAL\\_WWDG\\_ENABLE\\_IT](#)**Description:**

- Enables the WWDG early wakeup interrupt.

**Parameters:**

- \_\_HANDLE\_\_: WWDG handle
- \_\_INTERRUPT\_\_: specifies the interrupt to enable. This parameter can be one of the following values:
  - WWDG\_IT\_EWI: Early wakeup interrupt

**Return value:**

- None

**Notes:**

- Once enabled this interrupt cannot be disabled except by a system reset.

[\\_\\_HAL\\_WWDG\\_DISABLE\\_IT](#)**Description:**

- Disables the WWDG early wakeup interrupt.

**Parameters:**

- \_\_HANDLE\_\_: WWDG handle
- \_\_INTERRUPT\_\_: specifies the interrupt to disable. This parameter can be one of the following values:
  - WWDG\_IT\_EWI: Early wakeup interrupt

**Return value:**

- None

**Notes:**

- WARNING: This is a dummy macro for HAL code alignment. Once enabled this interrupt cannot be disabled except by a system reset.

`__HAL_WWDG_GET_FLAG`

**Description:**

- Gets the selected WWDG's flag status.

**Parameters:**

- `__HANDLE__`: WWDG handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

**Return value:**

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

`__HAL_WWDG_CLEAR_FLAG`

**Description:**

- Clears the WWDG's pending flags.

**Parameters:**

- `__HANDLE__`: WWDG handle
- `__FLAG__`: specifies the flag to clear. This parameter can be one of the following values:
  - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

**Return value:**

- None

`__HAL_WWDG_GET_IT_SOURCE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: WWDG Handle.
- `__INTERRUPT__`: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
  - `WWDG_IT_EWI`: Early Wakeup Interrupt

**Return value:**

- state: of `__INTERRUPT__` (TRUE or

---

FALSE).

***WWDG Flag definition***

WWDG\_FLAG\_EWIF Early wakeup interrupt flag

***WWDG Interrupt definition***

WWDG\_IT\_EWI Early wakeup interrupt

***WWDG Prescaler***

WWDG\_PRESCALER\_1 WWDG counter clock = (PCLK1/4096)/1

WWDG\_PRESCALER\_2 WWDG counter clock = (PCLK1/4096)/2

WWDG\_PRESCALER\_4 WWDG counter clock = (PCLK1/4096)/4

WWDG\_PRESCALER\_8 WWDG counter clock = (PCLK1/4096)/8

***WWDG Private Macros***

IS\_WWDG\_PRESCALER

IS\_WWDG\_WINDOW

IS\_WWDG\_COUNTER

## 47

# 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 STM32F2 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32F2 devices. To ensure compatibility between all devices and portability with other 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: `stm32f2xx_hal_conf.h`. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration, Ethernet parameter configuration...)

A template is provided in the HAL drivers folders (stm32f2xx\_hal\_conf\_template.c).

### Which header files should I include in my application to use the HAL drivers?

Only stm32f2xx\_hal.h file has to be included.

### What is the difference between stm32f2xx\_hal\_ppp.c/h and stm32f2xx\_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 (xx\_hal\_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (xx\_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 xx\_ll\_ppp.c?

These APIs cannot be used directly because they are internal and offer services to upper layer drivers. As an example xx\_ll\_fsmc.c/h driver is used by xx\_hal\_sram.c, xx\_hal\_nor.c, xx\_hal\_nand.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\_xx.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 xx\_hal\_msp.c. A template is provided in the HAL driver folders (xx\_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 `xx_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 xx\_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 xx\_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.

## 48      Revision history

Table 18: Document revision history

Date	Revision	Changes
19-Oct-2015	1	Initial release.

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved