

---

## Description of STM32F7xx 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 STM32CubeF7 for stm32f7 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>54</b>
<b>2</b>	<b>Overview of HAL drivers.....</b>	<b>56</b>
2.1	HAL and user-application files.....	56
2.1.1	HAL driver files .....	56
2.1.2	User-application files .....	57
2.2	HAL data structures .....	59
2.2.1	Peripheral handle structures .....	59
2.2.2	Initialization and configuration structure .....	60
2.2.3	Specific process structures .....	61
2.3	API classification .....	61
2.4	Devices supported by HAL drivers .....	62
2.5	HAL drivers rules.....	66
2.5.1	HAL API naming rules .....	66
2.5.2	HAL general naming rules.....	67
2.5.3	HAL interrupt handler and callback functions.....	68
2.6	HAL generic APIs.....	68
2.7	HAL extension APIs .....	70
2.7.1	HAL extension model overview .....	70
2.7.2	HAL extension model cases .....	70
2.8	File inclusion model.....	72
2.9	HAL common resources.....	73
2.10	HAL configuration.....	73
2.11	HAL system peripheral handling .....	74
2.11.1	Clock.....	74
2.11.2	GPIOs.....	75
2.11.3	Cortex NVIC and SysTick timer.....	76
2.11.4	PWR .....	76
2.11.5	EXTI.....	77
2.11.6	DMA.....	78
2.12	How to use HAL drivers .....	79
2.12.1	HAL usage models .....	79
2.12.2	HAL initialization .....	80
2.12.3	HAL IO operation process .....	83
2.12.4	Timeout and error management.....	86

<b>3</b>	<b>HAL System Driver .....</b>	<b>90</b>
3.1	HAL Firmware driver API description .....	90
3.1.1	How to use this driver .....	90
3.1.2	Initialization and de-initialization functions .....	90
3.1.3	HAL Control functions.....	90
3.1.4	HAL_Init.....	91
3.1.5	HAL_DeInit .....	91
3.1.6	HAL_MspInit .....	91
3.1.7	HAL_MspDeInit .....	92
3.1.8	HAL_InitTick .....	92
3.1.9	HAL_IncTick .....	92
3.1.10	HAL_GetTick .....	92
3.1.11	HAL_Delay .....	92
3.1.12	HAL_SuspendTick.....	93
3.1.13	HAL_ResumeTick.....	93
3.1.14	HAL_GetHalVersion .....	93
3.1.15	HAL_GetREVID .....	93
3.1.16	HAL_GetDEVID .....	93
3.1.17	HAL_DBGMCU_EnableDBGSleepMode .....	94
3.1.18	HAL_DBGMCU_DisableDBGSleepMode .....	94
3.1.19	HAL_DBGMCU_EnableDBGStopMode .....	94
3.1.20	HAL_DBGMCU_DisableDBGStopMode .....	94
3.1.21	HAL_DBGMCU_EnableDBGStandbyMode .....	94
3.1.22	HAL_DBGMCU_DisableDBGStandbyMode .....	94
3.1.23	HAL_EnableCompensationCell .....	94
3.1.24	HAL_DisableCompensationCell .....	95
3.1.25	HAL_EnableFMCMemorySwapping.....	95
3.1.26	HAL_DisableFMCMemorySwapping .....	95
3.2	HAL Firmware driver defines.....	95
3.2.1	HAL .....	95
<b>4</b>	<b>HAL ADC Generic Driver .....</b>	<b>98</b>
4.1	ADC Firmware driver registers structures .....	98
4.1.1	ADC_InitTypeDef.....	98
4.1.2	ADC_HandleTypeDef .....	99
4.1.3	ADC_ChannelConfTypeDef .....	99
4.1.4	ADC_AnalogWDGConfTypeDef.....	100
4.2	ADC Firmware driver API description.....	101

4.2.1	ADC Peripheral features.....	101
4.2.2	How to use this driver .....	101
4.2.3	Initialization and de-initialization functions .....	103
4.2.4	IO operation functions .....	103
4.2.5	Peripheral Control functions .....	104
4.2.6	Peripheral State and errors functions.....	104
4.2.7	HAL_ADC_Init .....	104
4.2.8	HAL_ADC_DeInit.....	105
4.2.9	HAL_ADC_MspInit .....	105
4.2.10	HAL_ADC_MspDeInit.....	105
4.2.11	HAL_ADC_Start .....	105
4.2.12	HAL_ADC_Stop.....	105
4.2.13	HAL_ADC_PollForConversion .....	106
4.2.14	HAL_ADC_PollForEvent .....	106
4.2.15	HAL_ADC_Start_IT .....	106
4.2.16	HAL_ADC_Stop_IT .....	106
4.2.17	HAL_ADC_IRQHandler.....	107
4.2.18	HAL_ADC_Start_DMA .....	107
4.2.19	HAL_ADC_Stop_DMA.....	107
4.2.20	HAL_ADC_GetValue .....	107
4.2.21	HAL_ADC_ConvCpltCallback .....	108
4.2.22	HAL_ADC_ConvHalfCpltCallback.....	108
4.2.23	HAL_ADC_LevelOutOfWindowCallback.....	108
4.2.24	HAL_ADC_ErrorCallback .....	108
4.2.25	HAL_ADC_ConfigChannel .....	108
4.2.26	HAL_ADC_AnalogWDGConfig .....	109
4.2.27	HAL_ADC_GetState.....	109
4.2.28	HAL_ADC_GetError .....	109
4.3	ADC Firmware driver defines .....	109
4.3.1	ADC .....	109
<b>5</b>	<b>HAL ADC Extension Driver .....</b>	<b>118</b>
5.1	ADCEX Firmware driver registers structures .....	118
5.1.1	ADC_InjectionConfTypeDef .....	118
5.1.2	ADC_MultiModeTypeDef.....	119
5.2	ADCEX Firmware driver API description .....	119
5.2.1	How to use this driver .....	119
5.2.2	Extended features functions .....	120
5.2.3	HAL_ADCEX_InjectedStart .....	121



5.2.4	HAL_ADCEx_InjectedStart_IT .....	121
5.2.5	HAL_ADCEx_InjectedStop .....	121
5.2.6	HAL_ADCEx_InjectedPollForConversion .....	122
5.2.7	HAL_ADCEx_InjectedStop_IT .....	122
5.2.8	HAL_ADCEx_InjectedGetValue .....	122
5.2.9	HAL_ADCEx_MultiModeStart_DMA .....	122
5.2.10	HAL_ADCEx_MultiModeStop_DMA .....	123
5.2.11	HAL_ADCEx_MultiModeGetValue .....	123
5.2.12	HAL_ADCEx_InjectedConvCpltCallback .....	123
5.2.13	HAL_ADCEx_InjectedConfigChannel .....	123
5.2.14	HAL_ADCEx_MultiModeConfigChannel .....	124
5.3	ADCEx Firmware driver defines .....	124
5.3.1	ADCEx .....	124
<b>6</b>	<b>HAL CAN Generic Driver .....</b>	<b>127</b>
6.1	CAN Firmware driver registers structures .....	127
6.1.1	CAN_InitTypeDef .....	127
6.1.2	CAN_FilterConfTypeDef .....	128
6.1.3	CanTxMsgTypeDef .....	129
6.1.4	CanRxMsgTypeDef .....	129
6.1.5	CAN_HandleTypeDef .....	130
6.2	CAN Firmware driver API description .....	131
6.2.1	How to use this driver .....	131
6.2.2	Initialization and de-initialization functions .....	132
6.2.3	IO operation functions .....	132
6.2.4	Peripheral State and Error functions .....	132
6.2.5	HAL_CAN_Init .....	133
6.2.6	HAL_CAN_ConfigFilter .....	133
6.2.7	HAL_CAN_DeInit .....	133
6.2.8	HAL_CAN_MspInit .....	133
6.2.9	HAL_CAN_MspDeInit .....	133
6.2.10	HAL_CAN_Transmit .....	134
6.2.11	HAL_CAN_Transmit_IT .....	134
6.2.12	HAL_CAN_Receive .....	134
6.2.13	HAL_CAN_Receive_IT .....	134
6.2.14	HAL_CAN_Sleep .....	134
6.2.15	HAL_CAN_WakeUp .....	135
6.2.16	HAL_CAN_IRQHandler .....	135
6.2.17	HAL_CAN_TxCpltCallback .....	135

6.2.18	HAL_CAN_RxCpltCallback .....	135
6.2.19	HAL_CAN_ErrorCallback .....	135
6.2.20	HAL_CAN_GetState .....	136
6.2.21	HAL_CAN_GetError .....	136
6.3	CAN Firmware driver defines .....	136
6.3.1	CAN .....	136
<b>7</b>	<b>HAL CEC Generic Driver .....</b>	<b>144</b>
7.1	CEC Firmware driver registers structures .....	144
7.1.1	CEC_InitTypeDef .....	144
7.1.2	CEC_HandleTypeDef .....	145
7.2	CEC Firmware driver API description .....	146
7.2.1	How to use this driver .....	146
7.2.2	Initialization and Configuration functions .....	146
7.2.3	IO operation functions .....	146
7.2.4	Peripheral Control function .....	147
7.2.5	HAL_CEC_Init .....	147
7.2.6	HAL_CEC_DeInit .....	147
7.2.7	HAL_CEC_MspInit .....	147
7.2.8	HAL_CEC_MspDeInit .....	147
7.2.9	HAL_CEC_Transmit .....	148
7.2.10	HAL_CEC_Receive .....	148
7.2.11	HAL_CEC_Transmit_IT .....	148
7.2.12	HAL_CEC_Receive_IT .....	149
7.2.13	HAL_CEC_GetReceivedFrameSize .....	149
7.2.14	HAL_CEC_IRQHandler .....	149
7.2.15	HAL_CEC_TxCpltCallback .....	149
7.2.16	HAL_CEC_RxCpltCallback .....	149
7.2.17	HAL_CEC_ErrorCallback .....	149
7.2.18	HAL_CEC_GetState .....	150
7.2.19	HAL_CEC_GetError .....	150
7.3	CEC Firmware driver defines .....	150
7.3.1	CEC .....	150
<b>8</b>	<b>HAL CORTEX Generic Driver .....</b>	<b>160</b>
8.1	CORTEX Firmware driver registers structures .....	160
8.1.1	MPU_Region_InitTypeDef .....	160
8.2	CORTEX Firmware driver API description .....	161
8.2.1	How to use this driver .....	161

8.2.2	Initialization and de-initialization functions .....	162
8.2.3	Peripheral Control functions .....	162
8.2.4	HAL_NVIC_SetPriorityGrouping .....	162
8.2.5	HAL_NVIC_SetPriority .....	162
8.2.6	HAL_NVIC_EnableIRQ .....	163
8.2.7	HAL_NVIC_DisableIRQ.....	163
8.2.8	HAL_NVIC_SystemReset.....	163
8.2.9	HAL_SYSTICK_Config.....	163
8.2.10	HAL_MPU_ConfigRegion.....	164
8.2.11	HAL_NVIC_GetPriorityGrouping .....	164
8.2.12	HAL_NVIC_GetPriority .....	164
8.2.13	HAL_NVIC_SetPendingIRQ.....	164
8.2.14	HAL_NVIC_GetPendingIRQ .....	165
8.2.15	HAL_NVIC_ClearPendingIRQ.....	165
8.2.16	HAL_NVIC_GetActive .....	165
8.2.17	HAL_SYSTICK_CLKSourceConfig .....	165
8.2.18	HAL_SYSTICK_IRQHandler .....	166
8.2.19	HAL_SYSTICK_Callback .....	166
8.3	CORTEX Firmware driver defines.....	166
8.3.1	CORTEX.....	166
<b>9</b>	<b>HAL CRC Generic Driver.....</b>	<b>170</b>
9.1	CRC Firmware driver registers structures .....	170
9.1.1	CRC_InitTypeDef .....	170
9.1.2	CRC_HandleTypeDef.....	171
9.2	CRC Firmware driver API description .....	171
9.2.1	CRC How to use this driver .....	171
9.2.2	Initialization and de-initialization functions .....	172
9.2.3	Peripheral Control functions .....	172
9.2.4	Peripheral State functions .....	172
9.2.5	HAL_CRC_Init.....	172
9.2.6	HAL_CRC_DeInit .....	172
9.2.7	HAL_CRC_MspInit .....	173
9.2.8	HAL_CRC_MspDeInit.....	173
9.2.9	HAL_CRC_Accumulate .....	173
9.2.10	HAL_CRC_Calculate.....	173
9.2.11	HAL_CRC_GetState.....	173
9.3	CRC Firmware driver defines.....	174
9.3.1	CRC.....	174

<b>10</b>	<b>HAL CRC Extension Driver .....</b>	<b>177</b>
10.1	CRCEX Firmware driver API description .....	177
10.1.1	CRC Extended features functions .....	177
10.1.2	HAL_CRCEX_Polynomial_Set .....	177
10.1.3	HAL_CRCEX_Input_Data_Reverse .....	177
10.1.4	HAL_CRCEX_Output_Data_Reverse.....	178
10.2	CRCEX Firmware driver defines.....	178
10.2.1	CRCEX.....	178
<b>11</b>	<b>HAL CRYP Generic Driver.....</b>	<b>180</b>
11.1	CRYP Firmware driver registers structures.....	180
11.1.1	CRYP_InitTypeDef .....	180
11.1.2	CRYP_HandleTypeDef.....	180
11.2	CRYP Firmware driver API description .....	181
11.2.1	How to use this driver .....	181
11.2.2	Initialization and de-initialization functions .....	182
11.2.3	AES processing functions .....	182
11.2.4	DES processing functions .....	183
11.2.5	TDES processing functions .....	183
11.2.6	DMA callback functions .....	184
11.2.7	CRYP IRQ handler management.....	184
11.2.8	Peripheral State functions .....	184
11.2.9	HAL_CRYP_Init.....	184
11.2.10	HAL_CRYP_DeInit .....	185
11.2.11	HAL_CRYP_MspInit .....	185
11.2.12	HAL_CRYP_MspDeInit .....	185
11.2.13	HAL_CRYP_AESECB_Encrypt.....	185
11.2.14	HAL_CRYP_AESCBC_Encrypt .....	185
11.2.15	HAL_CRYP_AESCTR_Encrypt.....	186
11.2.16	HAL_CRYP_AESECB_Decrypt .....	186
11.2.17	HAL_CRYP_AESCBC_Decrypt .....	186
11.2.18	HAL_CRYP_AESCTR_Decrypt .....	187
11.2.19	HAL_CRYP_AESECB_Encrypt_IT .....	187
11.2.20	HAL_CRYP_AESCBC_Encrypt_IT .....	187
11.2.21	HAL_CRYP_AESCTR_Encrypt_IT .....	188
11.2.22	HAL_CRYP_AESECB_Decrypt_IT .....	188
11.2.23	HAL_CRYP_AESCBC_Decrypt_IT .....	188
11.2.24	HAL_CRYP_AESCTR_Decrypt_IT .....	189
11.2.25	HAL_CRYP_AESECB_Encrypt_DMA.....	189

11.2.26	HAL_CRYP_AESCBC_Encrypt_DMA .....	189
11.2.27	HAL_CRYP_AESCTR_Encrypt_DMA.....	189
11.2.28	HAL_CRYP_AESECB_Decrypt_DMA .....	190
11.2.29	HAL_CRYP_AESCBC_Decrypt_DMA .....	190
11.2.30	HAL_CRYP_AESCTR_Decrypt_DMA .....	190
11.2.31	HAL_CRYP_DESECB_Encrypt .....	191
11.2.32	HAL_CRYP_DESECB_Decrypt .....	191
11.2.33	HAL_CRYP_DESCBC_Encrypt .....	191
11.2.34	HAL_CRYP_DESCBC_Decrypt .....	191
11.2.35	HAL_CRYP_DESECB_Encrypt_IT .....	192
11.2.36	HAL_CRYP_DESCBC_Encrypt_IT .....	192
11.2.37	HAL_CRYP_DESECB_Decrypt_IT .....	192
11.2.38	HAL_CRYP_DESCBC_Decrypt_IT .....	193
11.2.39	HAL_CRYP_DESECB_Encrypt_DMA .....	193
11.2.40	HAL_CRYP_DESCBC_Encrypt_DMA .....	193
11.2.41	HAL_CRYP_DESECB_Decrypt_DMA .....	193
11.2.42	HAL_CRYP_DESCBC_Decrypt_DMA .....	194
11.2.43	HAL_CRYP_TDESECB_Encrypt .....	194
11.2.44	HAL_CRYP_TDESECB_Decrypt .....	194
11.2.45	HAL_CRYP_TDESCBC_Encrypt .....	195
11.2.46	HAL_CRYP_TDESCBC_Decrypt .....	195
11.2.47	HAL_CRYP_TDESECB_Encrypt_IT .....	195
11.2.48	HAL_CRYP_TDESCBC_Encrypt_IT .....	196
11.2.49	HAL_CRYP_TDESECB_Decrypt_IT .....	196
11.2.50	HAL_CRYP_TDESCBC_Decrypt_IT .....	196
11.2.51	HAL_CRYP_TDESECB_Encrypt_DMA .....	196
11.2.52	HAL_CRYP_TDESCBC_Encrypt_DMA .....	197
11.2.53	HAL_CRYP_TDESECB_Decrypt_DMA .....	197
11.2.54	HAL_CRYP_TDESCBC_Decrypt_DMA .....	197
11.2.55	HAL_CRYP_InCpltCallback .....	198
11.2.56	HAL_CRYP_OutCpltCallback .....	198
11.2.57	HAL_CRYP_ErrorCallback .....	198
11.2.58	HAL_CRYP_IRQHandler .....	198
11.2.59	HAL_CRYP_GetState .....	198
11.3	CRYP Firmware driver defines .....	199
11.3.1	CRYP .....	199
<b>12</b>	<b>HAL CRYP Extension Driver .....</b>	<b>203</b>
12.1	CRYPEx Firmware driver API description .....	203

12.1.1	How to use this driver .....	203
12.1.2	Extended AES processing functions .....	204
12.1.3	CRYPEx IRQ handler management .....	204
12.1.4	HAL_CRYPEx_AESCCM_Encrypt .....	204
12.1.5	HAL_CRYPEx_AESGCM_Encrypt .....	205
12.1.6	HAL_CRYPEx_AESGCM_Decrypt .....	205
12.1.7	HAL_CRYPEx_AESGCM_Finish .....	205
12.1.8	HAL_CRYPEx_AESCCM_Finish .....	205
12.1.9	HAL_CRYPEx_AESCCM_Decrypt .....	206
12.1.10	HAL_CRYPEx_AESGCM_Encrypt_IT .....	206
12.1.11	HAL_CRYPEx_AESCCM_Encrypt_IT .....	206
12.1.12	HAL_CRYPEx_AESGCM_Decrypt_IT .....	207
12.1.13	HAL_CRYPEx_AESCCM_Decrypt_IT .....	207
12.1.14	HAL_CRYPEx_AESGCM_Encrypt_DMA .....	207
12.1.15	HAL_CRYPEx_AESCCM_Encrypt_DMA .....	207
12.1.16	HAL_CRYPEx_AESGCM_Decrypt_DMA .....	208
12.1.17	HAL_CRYPEx_AESCCM_Decrypt_DMA .....	208
12.1.18	HAL_CRYPEx_GCMCCM_IRQHandler .....	208
12.2	CRYPEx Firmware driver defines .....	209
12.2.1	CRYPEx .....	209
<b>13</b>	<b>HAL DAC Generic Driver .....</b>	<b>210</b>
13.1	DAC Firmware driver registers structures .....	210
13.1.1	DAC_HandleTypeDef .....	210
13.1.2	DAC_ChannelConfTypeDef .....	210
13.2	DAC Firmware driver API description .....	211
13.2.1	DAC Peripheral features .....	211
13.2.2	How to use this driver .....	212
13.2.3	Initialization and de-initialization functions .....	213
13.2.4	IO operation functions .....	213
13.2.5	Peripheral Control functions .....	213
13.2.6	Peripheral State and Errors functions .....	213
13.2.7	HAL_DAC_Init .....	214
13.2.8	HAL_DAC_DeInit .....	214
13.2.9	HAL_DAC_MspInit .....	214
13.2.10	HAL_DAC_MspDeInit .....	214
13.2.11	HAL_DAC_Start .....	214
13.2.12	HAL_DAC_Stop .....	215
13.2.13	HAL_DAC_Start_DMA .....	215

13.2.14	HAL_DAC_Stop_DMA.....	215
13.2.15	HAL_DAC_GetValue.....	216
13.2.16	HAL_DAC_IRQHandler.....	216
13.2.17	HAL_DAC_ConvCpltCallbackCh1.....	216
13.2.18	HAL_DAC_ConvHalfCpltCallbackCh1 .....	216
13.2.19	HAL_DAC_ErrorCallbackCh1 .....	217
13.2.20	HAL_DAC_DMAUnderrunCallbackCh1 .....	217
13.2.21	HAL_DAC_ConfigChannel .....	217
13.2.22	HAL_DAC_SetValue .....	217
13.2.23	HAL_DAC_GetState.....	218
13.2.24	HAL_DAC_GetError .....	218
13.2.25	HAL_DAC_IRQHandler.....	218
13.2.26	HAL_DAC_ConvCpltCallbackCh1.....	218
13.2.27	HAL_DAC_ConvHalfCpltCallbackCh1 .....	218
13.2.28	HAL_DAC_ErrorCallbackCh1 .....	219
13.2.29	HAL_DAC_DMAUnderrunCallbackCh1 .....	219
13.3	DAC Firmware driver defines .....	219
13.3.1	DAC .....	219
<b>14</b>	<b>HAL DAC Extension Driver .....</b>	<b>224</b>
14.1	DACEx Firmware driver API description .....	224
14.1.1	How to use this driver .....	224
14.1.2	Extended features functions .....	224
14.1.3	HAL_DACEx_DualGetValue .....	224
14.1.4	HAL_DACEx_TriangleWaveGenerate .....	224
14.1.5	HAL_DACEx_NoiseWaveGenerate .....	225
14.1.6	HAL_DACEx_DualSetValue.....	226
14.1.7	HAL_DACEx_ConvCpltCallbackCh2 .....	226
14.1.8	HAL_DACEx_ConvHalfCpltCallbackCh2 .....	226
14.1.9	HAL_DACEx_ErrorCallbackCh2 .....	227
14.1.10	HAL_DACEx_DMAUnderrunCallbackCh2 .....	227
14.2	DACEx Firmware driver defines .....	227
14.2.1	DACEx.....	227
<b>15</b>	<b>HAL DCMI Generic Driver .....</b>	<b>229</b>
15.1	DCMI Firmware driver registers structures.....	229
15.1.1	DCMI_HandleTypeDef .....	229
15.2	DCMI Firmware driver API description .....	229
15.2.1	How to use this driver .....	229

15.2.2	Initialization and Configuration functions.....	230
15.2.3	IO operation functions .....	230
15.2.4	Peripheral Control functions .....	231
15.2.5	Peripheral State and Errors functions .....	231
15.2.6	HAL_DCMI_Init.....	231
15.2.7	HAL_DCMI_DeInit .....	231
15.2.8	HAL_DCMI_MspInit.....	231
15.2.9	HAL_DCMI_MspDeInit .....	232
15.2.10	HAL_DCMI_Start_DMA.....	232
15.2.11	HAL_DCMI_Stop .....	232
15.2.12	HAL_DCMI_IRQHandler .....	232
15.2.13	HAL_DCMI_ErrorCallback .....	233
15.2.14	HAL_DCMI_LineEventCallback .....	233
15.2.15	HAL_DCMI_VsyncEventCallback .....	233
15.2.16	HAL_DCMI_FrameEventCallback.....	233
15.2.17	HAL_DCMI_ConfigCROP.....	233
15.2.18	HAL_DCMI_DisableCROP .....	234
15.2.19	HAL_DCMI_EnableCROP.....	234
15.2.20	HAL_DCMI_GetState .....	234
15.2.21	HAL_DCMI_GetError.....	234
15.3	DCMI Firmware driver defines.....	234
15.3.1	DCMI.....	234
<b>16</b>	<b>HAL DCMI Extension Driver.....</b>	<b>240</b>
16.1	DCMIEx Firmware driver registers structures.....	240
16.1.1	DCMI_CodesInitTypeDef.....	240
16.1.2	DCMI_InitTypeDef .....	240
16.2	DCMIEx Firmware driver API description .....	241
16.2.1	DCMI peripheral extension features.....	241
16.2.2	How to use this driver .....	241
16.2.3	Initialization and Configuration functions.....	241
16.2.4	HAL_DCMI_Init.....	241
16.3	DCMIEx Firmware driver defines .....	242
16.3.1	DCMIEx .....	242
<b>17</b>	<b>HAL DMA2D Generic Driver.....</b>	<b>243</b>
17.1	DMA2D Firmware driver registers structures .....	243
17.1.1	DMA2D_ColorTypeDef.....	243
17.1.2	DMA2D_CLUTCfgTypeDef .....	243
17.1.3	DMA2D_InitTypeDef.....	243



17.1.4	DMA2D_LayerCfgTypeDef .....	244
17.1.5	__DMA2D_HandleTypeDef .....	244
17.2	DMA2D Firmware driver API description .....	245
17.2.1	How to use this driver .....	245
17.2.2	Initialization and Configuration functions .....	246
17.2.3	IO operation functions .....	246
17.2.4	Peripheral Control functions .....	247
17.2.5	Peripheral State and Errors functions .....	247
17.2.6	HAL_DMA2D_Init .....	247
17.2.7	HAL_DMA2D_DeInit .....	248
17.2.8	HAL_DMA2D_MspInit .....	248
17.2.9	HAL_DMA2D_MspDeInit .....	248
17.2.10	HAL_DMA2D_Start .....	248
17.2.11	HAL_DMA2D_Start_IT .....	249
17.2.12	HAL_DMA2D_BlendingStart .....	249
17.2.13	HAL_DMA2D_BlendingStart_IT .....	249
17.2.14	HAL_DMA2D_Abort .....	250
17.2.15	HAL_DMA2D_Suspend .....	250
17.2.16	HAL_DMA2D_Resume .....	250
17.2.17	HAL_DMA2D_PollForTransfer .....	250
17.2.18	HAL_DMA2D_IRQHandler .....	251
17.2.19	HAL_DMA2D_ConfigLayer .....	251
17.2.20	HAL_DMA2D_ConfigCLUT .....	251
17.2.21	HAL_DMA2D_EnableCLUT .....	251
17.2.22	HAL_DMA2D_DisableCLUT .....	252
17.2.23	HAL_DMA2D_ProgramLineEvent .....	252
17.2.24	HAL_DMA2D_GetState .....	252
17.2.25	HAL_DMA2D_GetError .....	252
17.3	DMA2D Firmware driver defines .....	253
17.3.1	DMA2D .....	253
<b>18</b>	<b>HAL DMA Generic Driver .....</b>	<b>259</b>
18.1	DMA Firmware driver registers structures .....	259
18.1.1	DMA_InitTypeDef .....	259
18.1.2	__DMA_HandleTypeDef .....	260
18.2	DMA Firmware driver API description .....	261
18.2.1	How to use this driver .....	261
18.2.2	Initialization and de-initialization functions .....	262
18.2.3	IO operation functions .....	262

18.2.4	State and Errors functions .....	263
18.2.5	HAL_DMA_Init .....	263
18.2.6	HAL_DMA_DeInit .....	263
18.2.7	HAL_DMA_Start .....	263
18.2.8	HAL_DMA_Start_IT .....	263
18.2.9	HAL_DMA_Abort .....	264
18.2.10	HAL_DMA_PollForTransfer .....	264
18.2.11	HAL_DMA_IRQHandler .....	264
18.2.12	HAL_DMA_GetState .....	265
18.2.13	HAL_DMA_GetError .....	265
18.3	DMA Firmware driver defines .....	265
18.3.1	DMA .....	265
<b>19</b>	<b>HAL DMA Extension Driver .....</b>	<b>269</b>
19.1	DMAEx Firmware driver API description .....	269
19.1.1	How to use this driver .....	269
19.1.2	Extended features functions .....	269
19.1.3	HAL_DMAEx_MultiBufferStart .....	269
19.1.4	HAL_DMAEx_MultiBufferStart_IT .....	269
19.1.5	HAL_DMAEx_ChangeMemory .....	270
<b>20</b>	<b>HAL ETH Generic Driver .....</b>	<b>271</b>
20.1	ETH Firmware driver registers structures .....	271
20.1.1	ETH_InitTypeDef .....	271
20.1.2	ETH_MACInitTypeDef .....	271
20.1.3	ETH_DMAInitTypeDef .....	274
20.1.4	ETH_DMADescTypeDef .....	275
20.1.5	ETH_DMARxFramInfos .....	276
20.1.6	ETH_HandleTypeDef .....	277
20.2	ETH Firmware driver API description .....	277
20.2.1	How to use this driver .....	277
20.2.2	Initialization and de-initialization functions .....	278
20.2.3	IO operation functions .....	278
20.2.4	Peripheral Control functions .....	279
20.2.5	Peripheral State functions .....	279
20.2.6	HAL_ETH_Init .....	279
20.2.7	HAL_ETH_DeInit .....	279
20.2.8	HAL_ETH_DMATxDescListInit .....	279
20.2.9	HAL_ETH_DMARxDescListInit .....	280
20.2.10	HAL_ETH_MspltInit .....	280

20.2.11	HAL_ETH_MspDeInit .....	280
20.2.12	HAL_ETH_TransmitFrame .....	280
20.2.13	HAL_ETH_GetReceivedFrame .....	280
20.2.14	HAL_ETH_GetReceivedFrame_IT .....	281
20.2.15	HAL_ETH_IRQHandler .....	281
20.2.16	HAL_ETH_TxCpltCallback .....	281
20.2.17	HAL_ETH_RxCpltCallback .....	281
20.2.18	HAL_ETH_ErrorCallback .....	281
20.2.19	HAL_ETH_ReadPHYRegister .....	282
20.2.20	HAL_ETH_WritePHYRegister .....	282
20.2.21	HAL_ETH_Start .....	282
20.2.22	HAL_ETH_Stop .....	282
20.2.23	HAL_ETH_ConfigMAC .....	283
20.2.24	HAL_ETH_ConfigDMA .....	283
20.2.25	HAL_ETH_GetState .....	283
20.3	ETH Firmware driver defines .....	283
20.3.1	ETH .....	283
<b>21</b>	<b>HAL FLASH Generic Driver .....</b>	<b>315</b>
21.1	FLASH Firmware driver registers structures .....	315
21.1.1	FLASH_ProcessTypeDef .....	315
21.2	FLASH Firmware driver API description .....	315
21.2.1	FLASH peripheral features .....	315
21.2.2	How to use this driver .....	315
21.2.3	Programming operation functions .....	316
21.2.4	Peripheral Control functions .....	316
21.2.5	Peripheral Errors functions .....	317
21.2.6	HAL_FLASH_Program .....	317
21.2.7	HAL_FLASH_Program_IT .....	317
21.2.8	HAL_FLASH_IRQHandler .....	317
21.2.9	HAL_FLASH_EndOfOperationCallback .....	317
21.2.10	HAL_FLASH_OperationErrorCallback .....	318
21.2.11	HAL_FLASH_Unlock .....	318
21.2.12	HAL_FLASH_Lock .....	318
21.2.13	HAL_FLASH_OB_Unlock .....	318
21.2.14	HAL_FLASH_OB_Lock .....	318
21.2.15	HAL_FLASH_OB_Launch .....	318
21.2.16	HAL_FLASH_GetError .....	319
21.2.17	FLASH_WaitForLastOperation .....	319

21.3	FLASH Firmware driver defines .....	319
21.3.1	FLASH .....	319
<b>22</b>	<b>HAL FLASH Extension Driver .....</b>	<b>324</b>
22.1	FLASHEX Firmware driver registers structures .....	324
22.1.1	FLASH_EraseInitTypeDef .....	324
22.1.2	FLASH_OBProgramInitTypeDef .....	324
22.2	FLASHEX Firmware driver API description.....	325
22.2.1	Flash Extension features.....	325
22.2.2	How to use this driver .....	325
22.2.3	Extended programming operation functions .....	326
22.2.4	HAL_FLASHEx_Erase .....	326
22.2.5	HAL_FLASHEx_Erase_IT .....	326
22.2.6	HAL_FLASHEx_OBProgram.....	326
22.2.7	HAL_FLASHEx_OBGetConfig .....	326
22.3	FLASHEX Firmware driver defines .....	327
22.3.1	FLASHEX .....	327
<b>23</b>	<b>HAL GPIO Generic Driver.....</b>	<b>330</b>
23.1	GPIO Firmware driver registers structures .....	330
23.1.1	GPIO_InitTypeDef .....	330
23.2	GPIO Firmware driver API description .....	330
23.2.1	GPIO Peripheral features .....	330
23.2.2	How to use this driver .....	331
23.2.3	Initialization and de-initialization functions .....	331
23.2.4	IO operation functions .....	332
23.2.5	HAL_GPIO_Init.....	332
23.2.6	HAL_GPIO_DeInit .....	332
23.2.7	HAL_GPIO_ReadPin.....	332
23.2.8	HAL_GPIO_WritePin .....	332
23.2.9	HAL_GPIO_TogglePin .....	333
23.2.10	HAL_GPIO_LockPin.....	333
23.2.11	HAL_GPIO_EXTI_IRQHandler .....	333
23.2.12	HAL_GPIO_EXTI_Callback.....	333
23.3	GPIO Firmware driver defines.....	334
23.3.1	GPIO.....	334
<b>24</b>	<b>HAL GPIO Extension Driver .....</b>	<b>339</b>
24.1	GPIOEx Firmware driver defines.....	339
24.1.1	GPIOEx .....	339

<b>25</b>	<b>HAL HASH Generic Driver .....</b>	<b>340</b>
25.1	HASH Firmware driver registers structures .....	340
25.1.1	HASH_InitTypeDef .....	340
25.1.2	HASH_HandleTypeDef.....	340
25.2	HASH Firmware driver API description .....	341
25.2.1	How to use this driver .....	341
25.2.2	HASH processing using polling mode functions .....	342
25.2.3	HASH processing using interrupt mode functions .....	342
25.2.4	HASH processing using DMA mode functions .....	342
25.2.5	HMAC processing using polling mode functions .....	343
25.2.6	HMAC processing using DMA mode functions .....	343
25.2.7	Peripheral State functions .....	343
25.2.8	Initialization and de-initialization functions .....	343
25.2.9	HAL_HASH_MD5_Start .....	344
25.2.10	HAL_HASH_MD5_Accumulate .....	344
25.2.11	HAL_HASH_SHA1_Start.....	344
25.2.12	HAL_HASH_SHA1_Accumulate .....	345
25.2.13	HAL_HASH_MD5_Start_IT .....	345
25.2.14	HAL_HASH_SHA1_Start_IT .....	345
25.2.15	HAL_HASH_IRQHandler.....	346
25.2.16	HAL_HMAC_SHA1_Start.....	346
25.2.17	HAL_HMAC_MD5_Start.....	346
25.2.18	HAL_HASH_MD5_Start_DMA .....	346
25.2.19	HAL_HASH_MD5_Finish .....	347
25.2.20	HAL_HASH_SHA1_Start_DMA .....	347
25.2.21	HAL_HASH_SHA1_Finish .....	347
25.2.22	HAL_HASH_SHA1_Start_IT .....	347
25.2.23	HAL_HASH_MD5_Start_IT .....	348
25.2.24	HAL_HMAC_MD5_Start.....	348
25.2.25	HAL_HMAC_SHA1_Start.....	348
25.2.26	HAL_HASH_SHA1_Start_DMA .....	349
25.2.27	HAL_HASH_SHA1_Finish .....	349
25.2.28	HAL_HASH_MD5_Start_DMA .....	349
25.2.29	HAL_HASH_MD5_Finish .....	350
25.2.30	HAL_HMAC_MD5_Start_DMA.....	350
25.2.31	HAL_HMAC_SHA1_Start_DMA.....	350
25.2.32	HAL_HASH_GetState .....	351
25.2.33	HAL_HASH_IRQHandler.....	351

25.2.34	HAL_HASH_Init.....	351
25.2.35	HAL_HASH_DeInit .....	351
25.2.36	HAL_HASH_MspInit .....	351
25.2.37	HAL_HASH_MspDeInit .....	351
25.2.38	HAL_HASH_InCpltCallback .....	352
25.2.39	HAL_HASH_ErrorCallback.....	352
25.2.40	HAL_HASH_DgstCpltCallback.....	352
25.2.41	HAL_HASH_GetState .....	352
25.2.42	HAL_HASH_MspInit .....	352
25.2.43	HAL_HASH_MspDeInit .....	353
25.2.44	HAL_HASH_InCpltCallback .....	353
25.2.45	HAL_HASH_DgstCpltCallback.....	353
25.2.46	HAL_HASH_ErrorCallback.....	353
25.3	HASH Firmware driver defines.....	354
25.3.1	HASH.....	354
<b>26</b>	<b>HAL HASH Extension Driver.....</b>	<b>357</b>
26.1	HASHEX Firmware driver API description .....	357
26.1.1	How to use this driver .....	357
26.1.2	HASH processing using polling mode functions .....	358
26.1.3	HMAC processing using polling mode functions .....	358
26.1.4	HASH processing using interrupt functions.....	358
26.1.5	HASH processing using DMA functions .....	358
26.1.6	HMAC processing using DMA functions .....	359
26.1.7	HAL_HASHEX_SHA224_Start .....	359
26.1.8	HAL_HASHEX_SHA256_Start .....	359
26.1.9	HAL_HASHEX_SHA224_Accumulate .....	359
26.1.10	HAL_HASHEX_SHA256_Accumulate .....	360
26.1.11	HAL_HMACEx_SHA224_Start.....	360
26.1.12	HAL_HMACEx_SHA256_Start.....	360
26.1.13	HAL_HASHEX_SHA224_Start_IT .....	361
26.1.14	HAL_HASHEX_SHA256_Start_IT .....	361
26.1.15	HAL_HASHEX_IRQHandler .....	361
26.1.16	HAL_HASHEX_SHA224_Start_DMA .....	362
26.1.17	HAL_HASHEX_SHA224_Finish .....	362
26.1.18	HAL_HASHEX_SHA256_Start_DMA .....	362
26.1.19	HAL_HASHEX_SHA256_Finish .....	362
26.1.20	HAL_HMACEx_SHA224_Start_DMA.....	363
26.1.21	HAL_HMACEx_SHA256_Start_DMA.....	363

26.1.22	HAL_HASHEx_SHA224_Start .....	363
26.1.23	HAL_HASHEx_SHA256_Start .....	364
26.1.24	HAL_HASHEx_SHA224_Accumulate .....	364
26.1.25	HAL_HASHEx_SHA256_Accumulate .....	364
26.1.26	HAL_HMACEx_SHA224_Start.....	364
26.1.27	HAL_HMACEx_SHA256_Start.....	365
26.1.28	HAL_HASHEx_SHA224_Start_IT .....	365
26.1.29	HAL_HASHEx_SHA256_Start_IT .....	365
26.1.30	HAL_HASHEx_SHA224_Start_DMA .....	366
26.1.31	HAL_HASHEx_SHA224_Finish .....	366
26.1.32	HAL_HASHEx_SHA256_Start_DMA .....	366
26.1.33	HAL_HASHEx_SHA256_Finish .....	367
26.1.34	HAL_HMACEx_SHA224_Start_DMA.....	367
26.1.35	HAL_HMACEx_SHA256_Start_DMA.....	367
26.1.36	HAL_HASHEx_IRQHandler .....	368
<b>27</b>	<b>HAL HCD Generic Driver.....</b>	<b>369</b>
27.1	HCD Firmware driver registers structures .....	369
27.1.1	HCD_HandleTypeDef.....	369
27.2	HCD Firmware driver API description .....	369
27.2.1	How to use this driver .....	369
27.2.2	Initialization and de-initialization functions .....	370
27.2.3	IO operation functions .....	370
27.2.4	Peripheral Control functions .....	370
27.2.5	Peripheral State functions .....	370
27.2.6	HAL_HCD_Init.....	370
27.2.7	HAL_HCD_HC_Init.....	371
27.2.8	HAL_HCD_HC_Halt .....	371
27.2.9	HAL_HCD_DeInit .....	371
27.2.10	HAL_HCD_MspInit .....	371
27.2.11	HAL_HCD_MspDeInit.....	372
27.2.12	HAL_HCD_HC_SubmitRequest.....	372
27.2.13	HAL_HCD_IRQHandler.....	372
27.2.14	HAL_HCD_SOF_Callback .....	372
27.2.15	HAL_HCD_Connect_Callback .....	373
27.2.16	HAL_HCD_Disconnect_Callback .....	373
27.2.17	HAL_HCD_HC_NotifyURBChange_Callback .....	373
27.2.18	HAL_HCD_Start .....	373
27.2.19	HAL_HCD_Stop .....	373

27.2.20	HAL_HCD_ResetPort.....	374
27.2.21	HAL_HCD_GetState.....	374
27.2.22	HAL_HCD_HC_GetURBState.....	374
27.2.23	HAL_HCD_HC_GetXferCount .....	374
27.2.24	HAL_HCD_HC_GetState .....	374
27.2.25	HAL_HCD_GetCurrentFrame .....	375
27.2.26	HAL_HCD_GetCurrentSpeed .....	375
27.3	HCD Firmware driver defines.....	375
27.3.1	HCD.....	375
<b>28</b>	<b>HAL I2C Generic Driver .....</b>	<b>377</b>
28.1	I2C Firmware driver registers structures .....	377
28.1.1	I2C_InitTypeDef.....	377
28.1.2	I2C_HandleTypeDef .....	377
28.2	I2C Firmware driver API description.....	378
28.2.1	How to use this driver .....	378
28.2.2	Initialization and de-initialization functions .....	381
28.2.3	IO operation functions .....	381
28.2.4	Peripheral State and Errors functions .....	383
28.2.5	HAL_I2C_Init .....	383
28.2.6	HAL_I2C_DeInit.....	383
28.2.7	HAL_I2C_MspInit .....	383
28.2.8	HAL_I2C_MspDeInit.....	383
28.2.9	HAL_I2C_Master_Transmit.....	384
28.2.10	HAL_I2C_Master_Receive .....	384
28.2.11	HAL_I2C_Slave_Transmit.....	384
28.2.12	HAL_I2C_Slave_Receive .....	384
28.2.13	HAL_I2C_Master_Transmit_IT.....	385
28.2.14	HAL_I2C_Master_Receive_IT.....	385
28.2.15	HAL_I2C_Slave_Transmit_IT.....	385
28.2.16	HAL_I2C_Slave_Receive_IT.....	385
28.2.17	HAL_I2C_Master_Transmit_DMA.....	386
28.2.18	HAL_I2C_Master_Receive_DMA.....	386
28.2.19	HAL_I2C_Slave_Transmit_DMA .....	386
28.2.20	HAL_I2C_Slave_Receive_DMA.....	387
28.2.21	HAL_I2C_Mem_Write.....	387
28.2.22	HAL_I2C_Mem_Read .....	387
28.2.23	HAL_I2C_Mem_Write_IT .....	388
28.2.24	HAL_I2C_Mem_Read_IT .....	388



28.2.25	HAL_I2C_Mem_Write_DMA .....	388
28.2.26	HAL_I2C_Mem_Read_DMA .....	389
28.2.27	HAL_I2C_IsDeviceReady .....	389
28.2.28	HAL_I2C_EV_IRQHandler .....	389
28.2.29	HAL_I2C_ER_IRQHandler .....	389
28.2.30	HAL_I2C_MasterTxCpltCallback .....	390
28.2.31	HAL_I2C_MasterRxCpltCallback .....	390
28.2.32	HAL_I2C_SlaveTxCpltCallback .....	390
28.2.33	HAL_I2C_SlaveRxCpltCallback .....	390
28.2.34	HAL_I2C_MemTxCpltCallback .....	390
28.2.35	HAL_I2C_MemRxCpltCallback .....	391
28.2.36	HAL_I2C_ErrorCallback .....	391
28.2.37	HAL_I2C_GetState .....	391
28.2.38	HAL_I2C_GetError .....	391
28.3	I2C Firmware driver defines .....	391
28.3.1	I2C .....	391
<b>29</b>	<b>HAL I2C Extension Driver .....</b>	<b>398</b>
29.1	I2CEx Firmware driver API description .....	398
29.1.1	I2C peripheral Extended features .....	398
29.1.2	How to use this driver .....	398
29.1.3	Extended features functions .....	398
29.1.4	HAL_I2CEx_ConfigAnalogFilter .....	398
29.1.5	HAL_I2CEx_ConfigDigitalFilter .....	398
29.2	I2CEx Firmware driver defines .....	399
29.2.1	I2CEx .....	399
<b>30</b>	<b>HAL I2S Generic Driver .....</b>	<b>400</b>
30.1	I2S Firmware driver registers structures .....	400
30.1.1	I2S_InitTypeDef .....	400
30.1.2	I2S_HandleTypeDef .....	400
30.2	I2S Firmware driver API description .....	401
30.2.1	How to use this driver .....	401
30.2.2	Initialization and de-initialization functions .....	403
30.2.3	IO operation functions .....	403
30.2.4	Peripheral State and Errors functions .....	404
30.2.5	HAL_I2S_Init .....	404
30.2.6	HAL_I2S_DeInit .....	404
30.2.7	HAL_I2S_MspInit .....	405

30.2.8	HAL_I2S_MspDeInit .....	405
30.2.9	HAL_I2S_Transmit .....	405
30.2.10	HAL_I2S_Receive .....	405
30.2.11	HAL_I2S_Transmit_IT .....	406
30.2.12	HAL_I2S_Receive_IT .....	406
30.2.13	HAL_I2S_Transmit_DMA .....	407
30.2.14	HAL_I2S_Receive_DMA .....	407
30.2.15	HAL_I2S_DMAPause .....	407
30.2.16	HAL_I2S_DMAResume .....	408
30.2.17	HAL_I2S_DMAStop .....	408
30.2.18	HAL_I2S_IRQHandler .....	408
30.2.19	HAL_I2S_TxHalfCpltCallback .....	408
30.2.20	HAL_I2S_TxCpltCallback .....	408
30.2.21	HAL_I2S_RxHalfCpltCallback .....	409
30.2.22	HAL_I2S_RxCpltCallback .....	409
30.2.23	HAL_I2S_ErrorCallback .....	409
30.2.24	HAL_I2S_GetState .....	409
30.2.25	HAL_I2S_GetError .....	409
30.3	I2S Firmware driver defines .....	410
30.3.1	I2S .....	410
<b>31</b>	<b>HAL IRDA Generic Driver .....</b>	<b>414</b>
31.1	IRDA Firmware driver registers structures .....	414
31.1.1	IRDA_InitTypeDef .....	414
31.1.2	IRDA_HandleTypeDef .....	414
31.2	IRDA Firmware driver API description .....	415
31.2.1	How to use this driver .....	415
31.2.2	Initialization and Configuration functions .....	417
31.2.3	IO operation functions .....	417
31.2.4	Peripheral Control functions .....	418
31.2.5	HAL_IRDA_Init .....	418
31.2.6	HAL_IRDA_DeInit .....	418
31.2.7	HAL_IRDA_MspInit .....	419
31.2.8	HAL_IRDA_MspDeInit .....	419
31.2.9	HAL_IRDA_Transmit .....	419
31.2.10	HAL_IRDA_Receive .....	419
31.2.11	HAL_IRDA_Transmit_IT .....	420
31.2.12	HAL_IRDA_Receive_IT .....	420
31.2.13	HAL_IRDA_Transmit_DMA .....	420

31.2.14	HAL_IRDA_Receive_DMA .....	420
31.2.15	HAL_IRDA_DMABase .....	421
31.2.16	HAL_IRDA_DMAResume .....	421
31.2.17	HAL_IRDA_DMAStop .....	421
31.2.18	HAL_IRDA_IRQHandler .....	421
31.2.19	HAL_IRDA_TxHalfCpltCallback .....	421
31.2.20	HAL_IRDA_TxCpltCallback .....	422
31.2.21	HAL_IRDA_RxHalfCpltCallback .....	422
31.2.22	HAL_IRDA_RxCpltCallback .....	422
31.2.23	HAL_IRDA_ErrorCallback .....	422
31.2.24	HAL_IRDA_GetState .....	422
31.2.25	HAL_IRDA_GetError .....	423
31.3	IRDA Firmware driver defines .....	423
31.3.1	IRDA .....	423
<b>32</b>	<b>HAL IRDA Extension Driver .....</b>	<b>431</b>
32.1	IRDAEx Firmware driver defines .....	431
32.1.1	IRDAEx .....	431
<b>33</b>	<b>HAL IWDG Generic Driver .....</b>	<b>432</b>
33.1	IWDG Firmware driver registers structures .....	432
33.1.1	IWDG_InitTypeDef .....	432
33.1.2	IWDG_HandleTypeDef .....	432
33.2	IWDG Firmware driver API description .....	433
33.2.1	Initialization and de-initialization functions .....	433
33.2.2	IO operation functions .....	433
33.2.3	Peripheral State functions .....	433
33.2.4	HAL_IWDG_Init .....	433
33.2.5	HAL_IWDG_MsplInit .....	433
33.2.6	HAL_IWDG_Start .....	434
33.2.7	HAL_IWDG_Refresh .....	434
33.2.8	HAL_IWDG_GetState .....	434
33.3	IWDG Firmware driver defines .....	434
33.3.1	IWDG .....	434
<b>34</b>	<b>HAL LPTIM Generic Driver .....</b>	<b>438</b>
34.1	LPTIM Firmware driver registers structures .....	438
34.1.1	LPTIM_ClockConfigTypeDef .....	438
34.1.2	LPTIM_ULPClockConfigTypeDef .....	438
34.1.3	LPTIM_TriggerConfigTypeDef .....	438

34.1.4	LPTIM_InitTypeDef.....	439
34.1.5	LPTIM_HandleTypeDef.....	439
34.2	LPTIM Firmware driver API description.....	440
34.2.1	How to use this driver.....	440
34.2.2	Initialization and de-initialization functions .....	442
34.2.3	LPTIM Start Stop operation functions .....	442
34.2.4	LPTIM Read operation functions.....	443
34.2.5	LPTIM IRQ handler.....	443
34.2.6	Peripheral State functions .....	443
34.2.7	HAL_LPTIM_Init .....	443
34.2.8	HAL_LPTIM_DeInit.....	444
34.2.9	HAL_LPTIM_MspInit .....	444
34.2.10	HAL_LPTIM_MspDeInit.....	444
34.2.11	HAL_LPTIM_PWM_Start.....	444
34.2.12	HAL_LPTIM_PWM_Stop.....	444
34.2.13	HAL_LPTIM_PWM_Start_IT .....	445
34.2.14	HAL_LPTIM_PWM_Stop_IT .....	445
34.2.15	HAL_LPTIM_OnePulse_Start .....	445
34.2.16	HAL_LPTIM_OnePulse_Stop.....	445
34.2.17	HAL_LPTIM_OnePulse_Start_IT .....	445
34.2.18	HAL_LPTIM_OnePulse_Stop_IT .....	446
34.2.19	HAL_LPTIM_SetOnce_Start .....	446
34.2.20	HAL_LPTIM_SetOnce_Stop .....	446
34.2.21	HAL_LPTIM_SetOnce_Start_IT .....	446
34.2.22	HAL_LPTIM_SetOnce_Stop_IT .....	447
34.2.23	HAL_LPTIM_Encoder_Start.....	447
34.2.24	HAL_LPTIM_Encoder_Stop .....	447
34.2.25	HAL_LPTIM_Encoder_Start_IT.....	447
34.2.26	HAL_LPTIM_Encoder_Stop_IT .....	447
34.2.27	HAL_LPTIM_TimeOut_Start .....	448
34.2.28	HAL_LPTIM_TimeOut_Stop.....	448
34.2.29	HAL_LPTIM_TimeOut_Start_IT .....	448
34.2.30	HAL_LPTIM_TimeOut_Stop_IT .....	448
34.2.31	HAL_LPTIM_Counter_Start .....	448
34.2.32	HAL_LPTIM_Counter_Stop.....	449
34.2.33	HAL_LPTIM_Counter_Start_IT .....	449
34.2.34	HAL_LPTIM_Counter_Stop_IT .....	449
34.2.35	HAL_LPTIM_ReadCounter .....	449
34.2.36	HAL_LPTIM_ReadAutoReload .....	449

34.2.37	HAL_LPTIM_ReadCompare .....	450
34.2.38	HAL_LPTIM_IRQHandler .....	450
34.2.39	HAL_LPTIM_CompareMatchCallback .....	450
34.2.40	HAL_LPTIM_AutoReloadMatchCallback .....	450
34.2.41	HAL_LPTIM_TriggerCallback .....	450
34.2.42	HAL_LPTIM_CompareWriteCallback .....	450
34.2.43	HAL_LPTIM_AutoReloadWriteCallback .....	451
34.2.44	HAL_LPTIM_DirectionUpCallback .....	451
34.2.45	HAL_LPTIM_DirectionDownCallback .....	451
34.2.46	HAL_LPTIM_GetState .....	451
34.3	LPTIM Firmware driver defines .....	451
34.3.1	LPTIM .....	451
<b>35</b>	<b>HAL LTDC Generic Driver .....</b>	<b>458</b>
35.1	LTDC Firmware driver registers structures .....	458
35.1.1	LTDC_ColorTypeDef .....	458
35.1.2	LTDC_InitTypeDef .....	458
35.1.3	LTDC_LayerCfgTypeDef .....	459
35.1.4	LTDC_HandleTypeDef .....	460
35.2	LTDC Firmware driver API description .....	461
35.2.1	How to use this driver .....	461
35.2.2	Initialization and Configuration functions .....	462
35.2.3	IO operation functions .....	462
35.2.4	Peripheral Control functions .....	462
35.2.5	Peripheral State and Errors functions .....	463
35.2.6	HAL_LTDC_Init .....	463
35.2.7	HAL_LTDC_DeInit .....	463
35.2.8	HAL_LTDC_MspInit .....	463
35.2.9	HAL_LTDC_MspDeInit .....	464
35.2.10	HAL_LTDC_ErrorCallback .....	464
35.2.11	HAL_LTDC_LineEvenCallback .....	464
35.2.12	HAL_LTDC_IRQHandler .....	464
35.2.13	HAL_LTDC_ErrorCallback .....	464
35.2.14	HAL_LTDC_LineEvenCallback .....	464
35.2.15	HAL_LTDC_ConfigLayer .....	465
35.2.16	HAL_LTDC_ConfigColorKeying .....	465
35.2.17	HAL_LTDC_ConfigCLUT .....	465
35.2.18	HAL_LTDC_EnableColorKeying .....	466
35.2.19	HAL_LTDC_DisableColorKeying .....	466

35.2.20	HAL_LTDC_EnableCLUT .....	466
35.2.21	HAL_LTDC_DisableCLUT .....	466
35.2.22	HAL_LTDC_EnableDither .....	466
35.2.23	HAL_LTDC_DisableDither .....	467
35.2.24	HAL_LTDC_SetWindowSize .....	467
35.2.25	HAL_LTDC_SetWindowPosition .....	467
35.2.26	HAL_LTDC_SetPixelFormat .....	467
35.2.27	HAL_LTDC_SetAlpha .....	468
35.2.28	HAL_LTDC_SetAddress .....	468
35.2.29	HAL_LTDC_ProgramLineEvent .....	468
35.2.30	HAL_LTDC_GetState .....	468
35.2.31	HAL_LTDC_GetError .....	469
35.3	LTDC Firmware driver defines .....	469
35.3.1	LTDC .....	469
<b>36</b>	<b>HAL NAND Generic Driver .....</b>	<b>475</b>
36.1	NAND Firmware driver registers structures .....	475
36.1.1	NAND_IDTypeDef .....	475
36.1.2	NAND_AddressTypeDef .....	475
36.1.3	NAND_InfoTypeDef .....	475
36.1.4	NAND_HandleTypeDef .....	476
36.2	NAND Firmware driver API description .....	476
36.2.1	How to use this driver .....	476
36.2.2	NAND Initialization and de-initialization functions .....	477
36.2.3	NAND Input and Output functions .....	477
36.2.4	NAND Control functions .....	478
36.2.5	NAND State functions .....	478
36.2.6	HAL_NAND_Init .....	478
36.2.7	HAL_NAND_DeInit .....	478
36.2.8	HAL_NAND_MspInit .....	478
36.2.9	HAL_NAND_MspDeInit .....	479
36.2.10	HAL_NAND_IRQHandler .....	479
36.2.11	HAL_NAND_ITCallback .....	479
36.2.12	HAL_NAND_Read_ID .....	479
36.2.13	HAL_NAND_Reset .....	479
36.2.14	HAL_NAND_Read_Page .....	479
36.2.15	HAL_NAND_Write_Page .....	480
36.2.16	HAL_NAND_Read_SpareArea .....	480
36.2.17	HAL_NAND_Write_SpareArea .....	480

36.2.18	HAL_NAND_Erase_Block .....	481
36.2.19	HAL_NAND_Read_Status .....	481
36.2.20	HAL_NAND_Address_Inc .....	481
36.2.21	HAL_NAND_ECC_Enable .....	481
36.2.22	HAL_NAND_ECC_Disable .....	481
36.2.23	HAL_NAND_GetECC .....	482
36.2.24	HAL_NAND_GetState .....	482
36.2.25	HAL_NAND_Read_Status .....	482
36.3	NAND Firmware driver defines.....	482
36.3.1	NAND.....	482
<b>37</b>	<b>HAL NOR Generic Driver.....</b>	<b>485</b>
37.1	NOR Firmware driver registers structures .....	485
37.1.1	NOR_IDTypeDef .....	485
37.1.2	NOR_CFIDef .....	485
37.1.3	NOR_HandleTypeDef.....	485
37.2	NOR Firmware driver API description .....	486
37.2.1	How to use this driver .....	486
37.2.2	NOR Initialization and de_initialization functions .....	487
37.2.3	NOR Input and Output functions .....	487
37.2.4	NOR Control functions.....	487
37.2.5	NOR State functions.....	487
37.2.6	HAL_NOR_Init.....	487
37.2.7	HAL_NOR_DeInit .....	488
37.2.8	HAL_NOR_MspInit .....	488
37.2.9	HAL_NOR_MspDeInit .....	488
37.2.10	HAL_NOR_MspWait.....	488
37.2.11	HAL_NOR_Read_ID .....	488
37.2.12	HAL_NOR_ReturnToReadMode .....	489
37.2.13	HAL_NOR_Read .....	489
37.2.14	HAL_NOR_Program.....	489
37.2.15	HAL_NOR_ReadBuffer .....	489
37.2.16	HAL_NOR_ProgramBuffer .....	490
37.2.17	HAL_NOR_Erase_Block .....	490
37.2.18	HAL_NOR_Erase_Chip.....	490
37.2.19	HAL_NOR_Read_CFI .....	490
37.2.20	HAL_NOR_WriteOperation_Enable .....	491
37.2.21	HAL_NOR_WriteOperation_Disable .....	491
37.2.22	HAL_NOR_GetState .....	491

37.2.23	HAL_NOR_GetStatus.....	491
37.3	NOR Firmware driver defines.....	491
37.3.1	NOR.....	491
<b>38</b>	<b>HAL PCD Generic Driver .....</b>	<b>494</b>
38.1	PCD Firmware driver registers structures .....	494
38.1.1	PCD_HandleTypeDef .....	494
38.2	PCD Firmware driver API description.....	495
38.2.1	How to use this driver .....	495
38.2.2	Initialization and de-initialization functions .....	495
38.2.3	IO operation functions .....	495
38.2.4	Peripheral Control functions .....	495
38.2.5	Peripheral State functions .....	496
38.2.6	HAL_PCD_Init .....	496
38.2.7	HAL_PCD_DeInit.....	496
38.2.8	HAL_PCD_MspInit .....	496
38.2.9	HAL_PCD_MspDeInit.....	496
38.2.10	HAL_PCD_Start .....	497
38.2.11	HAL_PCD_Stop.....	497
38.2.12	HAL_PCD_IRQHandler.....	497
38.2.13	HAL_PCD_DataOutStageCallback .....	497
38.2.14	HAL_PCD_DataInStageCallback .....	497
38.2.15	HAL_PCD_SetupStageCallback .....	498
38.2.16	HAL_PCD_SOFCallback.....	498
38.2.17	HAL_PCD_ResetCallback.....	498
38.2.18	HAL_PCD_SuspendCallback.....	498
38.2.19	HAL_PCD_ResumeCallback.....	498
38.2.20	HAL_PCD_ISOOUTIncompleteCallback.....	498
38.2.21	HAL_PCD_ISOINIncompleteCallback.....	499
38.2.22	HAL_PCD_ConnectCallback.....	499
38.2.23	HAL_PCD_DisconnectCallback .....	499
38.2.24	HAL_PCD_DevConnect .....	499
38.2.25	HAL_PCD_DevDisconnect.....	499
38.2.26	HAL_PCD_SetAddress .....	499
38.2.27	HAL_PCD_EP_Open .....	500
38.2.28	HAL_PCD_EP_Close .....	500
38.2.29	HAL_PCD_EP_Receive .....	500
38.2.30	HAL_PCD_EP_GetRxCount .....	500
38.2.31	HAL_PCD_EP_Transmit .....	501



38.2.32	HAL_PCD_EP_SetStall.....	501
38.2.33	HAL_PCD_EP_ClrStall.....	501
38.2.34	HAL_PCD_EP_Flush .....	501
38.2.35	HAL_PCD_ActivateRemoteWakeup .....	501
38.2.36	HAL_PCD_DeActivateRemoteWakeup.....	502
38.2.37	HAL_PCD_GetState.....	502
38.3	PCD Firmware driver defines .....	502
38.3.1	PCD .....	502
<b>39</b>	<b>HAL PCD Extension Driver .....</b>	<b>505</b>
39.1	PCDEx Firmware driver API description .....	505
39.1.1	Extended features functions .....	505
39.1.2	HAL_PCDEx_SetTxFiFo .....	505
39.1.3	HAL_PCDEx_SetRxFiFo.....	505
39.1.4	HAL_PCDEx_ActivateLPM .....	505
39.1.5	HAL_PCDEx_DeActivateLPM .....	505
39.1.6	HAL_PCDEx_LPM_Callback .....	506
<b>40</b>	<b>HAL PWR Generic Driver .....</b>	<b>507</b>
40.1	PWR Firmware driver registers structures .....	507
40.1.1	PWR_PVDTypeDef .....	507
40.2	PWR Firmware driver API description.....	507
40.2.1	Initialization and de-initialization functions .....	507
40.2.2	Peripheral Control functions .....	507
40.2.3	HAL_PWR_DeInit.....	509
40.2.4	HAL_PWR_EnableBkUpAccess .....	509
40.2.5	HAL_PWR_DisableBkUpAccess.....	510
40.2.6	HAL_PWR_ConfigPVD .....	510
40.2.7	HAL_PWR_EnablePVD.....	510
40.2.8	HAL_PWR_DisablePVD.....	510
40.2.9	HAL_PWR_EnableWakeUpPin .....	510
40.2.10	HAL_PWR_DisableWakeUpPin .....	511
40.2.11	HAL_PWR_EnterSLEEPMode.....	511
40.2.12	HAL_PWR_EnterSTOPMode.....	512
40.2.13	HAL_PWR_EnterSTANDBYMode .....	512
40.2.14	HAL_PWR_PVD_IRQHandler.....	512
40.2.15	HAL_PWR_PVDCallback.....	512
40.2.16	HAL_PWR_EnableSleepOnExit.....	513
40.2.17	HAL_PWR_DisableSleepOnExit .....	513

40.2.18	HAL_PWR_EnableSEVOnPend .....	513
40.2.19	HAL_PWR_DisableSEVOnPend.....	513
40.3	PWR Firmware driver defines .....	514
40.3.1	PWR .....	514
<b>41</b>	<b>HAL PWR Extension Driver .....</b>	<b>519</b>
41.1	PWREx Firmware driver API description.....	519
41.1.1	Peripheral extended features functions.....	519
41.1.2	HAL_PWREx_EnableBkUpReg .....	520
41.1.3	HAL_PWREx_DisableBkUpReg .....	520
41.1.4	HAL_PWREx_EnableFlashPowerDown .....	520
41.1.5	HAL_PWREx_DisableFlashPowerDown.....	520
41.1.6	HAL_PWREx_EnableMainRegulatorLowVoltage .....	520
41.1.7	HAL_PWREx_DisableMainRegulatorLowVoltage .....	521
41.1.8	HAL_PWREx_EnableLowRegulatorLowVoltage .....	521
41.1.9	HAL_PWREx_DisableLowRegulatorLowVoltage.....	521
41.1.10	HAL_PWREx_EnableOverDrive .....	521
41.1.11	HAL_PWREx_DisableOverDrive.....	521
41.1.12	HAL_PWREx_EnterUnderDriveSTOPMode .....	522
41.1.13	HAL_PWREx_GetVoltageRange .....	522
41.1.14	HAL_PWREx_ControlVoltageScaling .....	523
41.2	PWREx Firmware driver defines .....	523
41.2.1	PWREx.....	523
<b>42</b>	<b>HAL QSPI Generic Driver .....</b>	<b>527</b>
42.1	QSPI Firmware driver registers structures .....	527
42.1.1	QSPI_InitTypeDef.....	527
42.1.2	QSPI_HandleTypeDef .....	527
42.1.3	QSPI_CommandTypeDef.....	528
42.1.4	QSPI_AutoPollingTypeDef .....	529
42.1.5	QSPI_MemoryMappedTypeDef .....	529
42.2	QSPI Firmware driver API description.....	529
42.2.1	How to use this driver .....	529
42.2.2	Initialization and Configuration functions.....	532
42.2.3	IO operation functions .....	532
42.2.4	Peripheral Control and State functions.....	532
42.2.5	HAL_QSPI_Init .....	533
42.2.6	HAL_QSPI_DeInit.....	533
42.2.7	HAL_QSPI_MspInit .....	533
42.2.8	HAL_QSPI_MspDeInit.....	533

42.2.9	HAL_QSPI_IRQHandler .....	533
42.2.10	HAL_QSPI_Command .....	534
42.2.11	HAL_QSPI_Command_IT .....	534
42.2.12	HAL_QSPI_Transmit .....	534
42.2.13	HAL_QSPI_Receive .....	534
42.2.14	HAL_QSPI_Transmit_IT .....	535
42.2.15	HAL_QSPI_Receive_IT .....	535
42.2.16	HAL_QSPI_Transmit_DMA .....	535
42.2.17	HAL_QSPI_Receive_DMA .....	535
42.2.18	HAL_QSPI_AutoPolling .....	536
42.2.19	HAL_QSPI_AutoPolling_IT .....	536
42.2.20	HAL_QSPI_MemoryMapped .....	536
42.2.21	HAL_QSPI_ErrorCallback .....	537
42.2.22	HAL_QSPI_CmdCpltCallback .....	537
42.2.23	HAL_QSPI_RxCpltCallback .....	537
42.2.24	HAL_QSPI_TxCpltCallback .....	537
42.2.25	HAL_QSPI_RxHalfCpltCallback .....	537
42.2.26	HAL_QSPI_TxHalfCpltCallback .....	537
42.2.27	HAL_QSPI_FifoThresholdCallback .....	538
42.2.28	HAL_QSPI_StatusMatchCallback .....	538
42.2.29	HAL_QSPI_TimeOutCallback .....	538
42.2.30	HAL_QSPI_GetState .....	538
42.2.31	HAL_QSPI_GetError .....	538
42.2.32	HAL_QSPI_Abort .....	538
42.2.33	HAL_QSPI_SetTimeout .....	539
42.2.34	HAL_QSPI_ErrorCallback .....	539
42.2.35	HAL_QSPI_FifoThresholdCallback .....	539
42.2.36	HAL_QSPI_CmdCpltCallback .....	539
42.2.37	HAL_QSPI_RxCpltCallback .....	539
42.2.38	HAL_QSPI_TxCpltCallback .....	539
42.2.39	HAL_QSPI_RxHalfCpltCallback .....	540
42.2.40	HAL_QSPI_TxHalfCpltCallback .....	540
42.2.41	HAL_QSPI_StatusMatchCallback .....	540
42.2.42	HAL_QSPI_TimeOutCallback .....	540
42.2.43	HAL_QSPI_GetState .....	540
42.2.44	HAL_QSPI_GetError .....	541
42.2.45	HAL_QSPI_Abort .....	541
42.2.46	HAL_QSPI_SetTimeout .....	541

42.3	QSPI Firmware driver defines .....	541
42.3.1	QSPI .....	541
<b>43</b>	<b>HAL RCC Generic Driver .....</b>	<b>549</b>
43.1	RCC Firmware driver registers structures .....	549
43.1.1	RCC_PLLInitTypeDef .....	549
43.1.2	RCC_OscInitTypeDef .....	549
43.1.3	RCC_ClkInitTypeDef .....	550
43.2	RCC Firmware driver API description .....	551
43.2.1	RCC specific features .....	551
43.2.2	RCC Limitations.....	551
43.2.3	Initialization and de-initialization functions .....	551
43.2.4	Peripheral Control functions .....	552
43.2.5	HAL_RCC_DeInit .....	552
43.2.6	HAL_RCC_OscConfig .....	553
43.2.7	HAL_RCC_ClockConfig .....	553
43.2.8	HAL_RCC_MCOConfig.....	554
43.2.9	HAL_RCC_EnableCSS .....	554
43.2.10	HAL_RCC_DisableCSS .....	555
43.2.11	HAL_RCC_GetSysClockFreq .....	555
43.2.12	HAL_RCC_GetHCLKFreq .....	555
43.2.13	HAL_RCC_GetPCLK1Freq .....	555
43.2.14	HAL_RCC_GetPCLK2Freq .....	556
43.2.15	HAL_RCC_GetOscConfig .....	556
43.2.16	HAL_RCC_GetClockConfig .....	556
43.2.17	HAL_RCC_NMI_IRQHandler .....	556
43.2.18	HAL_RCC_CSSCallback.....	556
43.3	RCC Firmware driver defines .....	557
43.3.1	RCC .....	557
<b>44</b>	<b>HAL RCC Extension Driver .....</b>	<b>580</b>
44.1	RCCEX Firmware driver registers structures .....	580
44.1.1	RCC_PLLI2SInitTypeDef.....	580
44.1.2	RCC_PLLSAIInitTypeDef .....	580
44.1.3	RCC_PeriphCLKInitTypeDef .....	581
44.2	RCCEX Firmware driver API description .....	583
44.2.1	Extended Peripheral Control functions .....	583
44.2.2	HAL_RCCEX_PeriphCLKConfig.....	583
44.2.3	HAL_RCCEX_GetPeriphCLKConfig.....	584

44.2.4	HAL_RCCEX_GetPeriphCLKFreq .....	584
44.3	RCCEX Firmware driver defines .....	584
44.3.1	RCCEX .....	584
<b>45</b>	<b>HAL RNG Generic Driver .....</b>	<b>622</b>
45.1	RNG Firmware driver registers structures .....	622
45.1.1	RNG_HandleTypeDef .....	622
45.2	RNG Firmware driver API description .....	622
45.2.1	How to use this driver .....	622
45.2.2	Initialization and de-initialization functions .....	622
45.2.3	Peripheral Control functions .....	623
45.2.4	Peripheral State functions .....	623
45.2.5	HAL_RNG_Init .....	623
45.2.6	HAL_RNG_DeInit .....	623
45.2.7	HAL_RNG_MspInit .....	623
45.2.8	HAL_RNG_MspDeInit .....	624
45.2.9	HAL_RNG_GenerateRandomNumber .....	624
45.2.10	HAL_RNG_GenerateRandomNumber_IT .....	624
45.2.11	HAL_RNG_IRQHandler .....	624
45.2.12	HAL_RNG_GetRandomNumber .....	625
45.2.13	HAL_RNG_GetRandomNumber_IT .....	625
45.2.14	HAL_RNG_ReadLastRandomNumber .....	625
45.2.15	HAL_RNG_ReadyDataCallback .....	625
45.2.16	HAL_RNG_ErrorCallback .....	626
45.2.17	HAL_RNG_GetState .....	626
45.3	RNG Firmware driver defines .....	626
45.3.1	RNG .....	626
<b>46</b>	<b>HAL RTC Generic Driver .....</b>	<b>629</b>
46.1	RTC Firmware driver registers structures .....	629
46.1.1	RTC_InitTypeDef .....	629
46.1.2	RTC_TimeTypeDef .....	629
46.1.3	RTC_DateTypeDef .....	630
46.1.4	RTC_AlarmTypeDef .....	631
46.1.5	RTC_HandleTypeDef .....	631
46.2	RTC Firmware driver API description .....	632
46.2.1	Backup Domain Operating Condition .....	632
46.2.2	Backup Domain Reset .....	632
46.2.3	Backup Domain Access .....	632

46.2.4	How to use this driver .....	633
46.2.5	RTC and low power modes .....	633
46.2.6	Initialization and de-initialization functions .....	633
46.2.7	RTC Time and Date functions .....	634
46.2.8	RTC Alarm functions .....	634
46.2.9	Peripheral Control functions .....	634
46.2.10	Peripheral State functions .....	634
46.2.11	HAL_RTC_Init .....	634
46.2.12	HAL_RTC_DeInit .....	635
46.2.13	HAL_RTC_MspInit .....	635
46.2.14	HAL_RTC_MspDeInit .....	635
46.2.15	HAL_RTC_SetTime .....	635
46.2.16	HAL_RTC_GetTime .....	636
46.2.17	HAL_RTC_SetDate .....	636
46.2.18	HAL_RTC_GetDate .....	636
46.2.19	HAL_RTC_SetAlarm .....	636
46.2.20	HAL_RTC_SetAlarm_IT .....	637
46.2.21	HAL_RTC_DeactivateAlarm .....	637
46.2.22	HAL_RTC_GetAlarm .....	637
46.2.23	HAL_RTC_AlarmIRQHandler .....	638
46.2.24	HAL_RTC_AlarmAEventCallback .....	638
46.2.25	HAL_RTC_PollForAlarmAEvent .....	638
46.2.26	HAL_RTC_WaitForSynchro .....	638
46.2.27	HAL_RTC_GetState .....	639
46.2.28	HAL_RTC_Init .....	639
46.2.29	HAL_RTC_DeInit .....	639
46.2.30	HAL_RTC_MspInit .....	639
46.2.31	HAL_RTC_MspDeInit .....	639
46.2.32	HAL_RTC_SetTime .....	640
46.2.33	HAL_RTC_GetTime .....	640
46.2.34	HAL_RTC_SetDate .....	640
46.2.35	HAL_RTC_GetDate .....	641
46.2.36	HAL_RTC_SetAlarm .....	641
46.2.37	HAL_RTC_SetAlarm_IT .....	641
46.2.38	HAL_RTC_DeactivateAlarm .....	641
46.2.39	HAL_RTC_GetAlarm .....	642
46.2.40	HAL_RTC_AlarmIRQHandler .....	642
46.2.41	HAL_RTC_PollForAlarmAEvent .....	642
46.2.42	HAL_RTC_AlarmAEventCallback .....	642

46.2.43	HAL_RTC_WaitForSynchro .....	643
46.2.44	HAL_RTC_GetState .....	643
46.3	RTC Firmware driver defines .....	643
46.3.1	RTC .....	643
<b>47</b>	<b>HAL RTC Extension Driver .....</b>	<b>654</b>
47.1	RTCEX Firmware driver registers structures .....	654
47.1.1	RTC_TamperTypeDef .....	654
47.2	RTCEX Firmware driver API description .....	655
47.2.1	How to use this driver .....	655
47.2.2	RTC TimeStamp and Tamper functions .....	656
47.2.3	RTC Wake-up functions .....	656
47.2.4	Extension Peripheral Control functions .....	656
47.2.5	Extended features functions .....	657
47.2.6	HAL_RTCEX_SetTimeStamp .....	657
47.2.7	HAL_RTCEX_SetTimeStamp_IT .....	657
47.2.8	HAL_RTCEX_DeactivateTimeStamp .....	658
47.2.9	HAL_RTCEX_SetInternalTimeStamp .....	658
47.2.10	HAL_RTCEX_DeactivateInternalTimeStamp .....	658
47.2.11	HAL_RTCEX_GetTimeStamp .....	659
47.2.12	HAL_RTCEX_SetTamper .....	659
47.2.13	HAL_RTCEX_SetTamper_IT .....	659
47.2.14	HAL_RTCEX_DeactivateTamper .....	659
47.2.15	HAL_RTCEX_TamperTimeStampIRQHandler .....	660
47.2.16	HAL_RTCEX_TimeStampEventCallback .....	660
47.2.17	HAL_RTCEX_Tamper1EventCallback .....	660
47.2.18	HAL_RTCEX_Tamper2EventCallback .....	660
47.2.19	HAL_RTCEX_Tamper3EventCallback .....	660
47.2.20	HAL_RTCEX_PollForTimeStampEvent .....	661
47.2.21	HAL_RTCEX_PollForTamper1Event .....	661
47.2.22	HAL_RTCEX_PollForTamper2Event .....	661
47.2.23	HAL_RTCEX_PollForTamper3Event .....	661
47.2.24	HAL_RTCEX_SetWakeUpTimer .....	661
47.2.25	HAL_RTCEX_SetWakeUpTimer_IT .....	662
47.2.26	HAL_RTCEX_DeactivateWakeUpTimer .....	662
47.2.27	HAL_RTCEX_GetWakeUpTimer .....	662
47.2.28	HAL_RTCEX_WakeUpTimerIRQHandler .....	662
47.2.29	HAL_RTCEX_WakeUpTimerEventCallback .....	663
47.2.30	HAL_RTCEX_PollForWakeUpTimerEvent .....	663

47.2.31	HAL_RTCEx_BKUPWrite .....	663
47.2.32	HAL_RTCEx_BKUPRead .....	663
47.2.33	HAL_RTCEx_SetSmoothCalib .....	664
47.2.34	HAL_RTCEx_SetSynchroShift .....	664
47.2.35	HAL_RTCEx_SetCalibrationOutPut .....	665
47.2.36	HAL_RTCEx_DeactivateCalibrationOutPut .....	665
47.2.37	HAL_RTCEx_SetRefClock .....	665
47.2.38	HAL_RTCEx_DeactivateRefClock .....	665
47.2.39	HAL_RTCEx_EnableBypassShadow .....	665
47.2.40	HAL_RTCEx_DisableBypassShadow .....	666
47.2.41	HAL_RTCEx_AlarmBEventCallback .....	666
47.2.42	HAL_RTCEx_PollForAlarmBEvent .....	666
47.2.43	HAL_RTCEx_SetTimeStamp .....	666
47.2.44	HAL_RTCEx_SetTimeStamp_IT .....	667
47.2.45	HAL_RTCEx_DeactivateTimeStamp .....	667
47.2.46	HAL_RTCEx_SetInternalTimeStamp .....	667
47.2.47	HAL_RTCEx_DeactivateInternalTimeStamp .....	668
47.2.48	HAL_RTCEx_GetTimeStamp .....	668
47.2.49	HAL_RTCEx_SetTamper .....	668
47.2.50	HAL_RTCEx_SetTamper_IT .....	669
47.2.51	HAL_RTCEx_DeactivateTamper .....	669
47.2.52	HAL_RTCEx_TamperTimeStampIRQHandler .....	669
47.2.53	HAL_RTCEx_Tamper1EventCallback .....	669
47.2.54	HAL_RTCEx_Tamper2EventCallback .....	669
47.2.55	HAL_RTCEx_Tamper3EventCallback .....	670
47.2.56	HAL_RTCEx_TimeStampEventCallback .....	670
47.2.57	HAL_RTCEx_PollForTimeStampEvent .....	670
47.2.58	HAL_RTCEx_PollForTamper1Event .....	670
47.2.59	HAL_RTCEx_PollForTamper2Event .....	670
47.2.60	HAL_RTCEx_PollForTamper3Event .....	671
47.2.61	HAL_RTCEx_SetWakeUpTimer .....	671
47.2.62	HAL_RTCEx_SetWakeUpTimer_IT .....	671
47.2.63	HAL_RTCEx_DeactivateWakeUpTimer .....	671
47.2.64	HAL_RTCEx_GetWakeUpTimer .....	672
47.2.65	HAL_RTCEx_WakeUpTimerIRQHandler .....	672
47.2.66	HAL_RTCEx_WakeUpTimerEventCallback .....	672
47.2.67	HAL_RTCEx_PollForWakeUpTimerEvent .....	672
47.2.68	HAL_RTCEx_BKUPWrite .....	672
47.2.69	HAL_RTCEx_BKUPRead .....	673



47.2.70	HAL_RTCEx_SetSmoothCalib .....	673
47.2.71	HAL_RTCEx_SetSynchroShift .....	673
47.2.72	HAL_RTCEx_SetCalibrationOutPut .....	674
47.2.73	HAL_RTCEx_DeactivateCalibrationOutPut .....	674
47.2.74	HAL_RTCEx_SetRefClock .....	674
47.2.75	HAL_RTCEx_DeactivateRefClock .....	675
47.2.76	HAL_RTCEx_EnableBypassShadow .....	675
47.2.77	HAL_RTCEx_DisableBypassShadow .....	675
47.2.78	HAL_RTCEx_AlarmBEventCallback .....	675
47.2.79	HAL_RTCEx_PollForAlarmBEvent .....	675
47.3	RTCEx Firmware driver defines .....	676
47.3.1	RTCEx .....	676
<b>48</b>	<b>HAL SAI Generic Driver .....</b>	<b>697</b>
48.1	SAI Firmware driver registers structures .....	697
48.1.1	SAI_InitTypeDef .....	697
48.1.2	SAI_FrameInitTypeDef .....	698
48.1.3	SAI_SlotInitTypeDef .....	699
48.1.4	__SAI_HandleTypeDef .....	699
48.2	SAI Firmware driver API description .....	700
48.2.1	How to use this driver .....	700
48.2.2	Initialization and de-initialization functions .....	702
48.2.3	IO operation functions .....	703
48.2.4	Peripheral State and Errors functions .....	704
48.2.5	HAL_SAI_InitProtocol .....	704
48.2.6	HAL_SAI_Init .....	704
48.2.7	HAL_SAI_DeInit .....	704
48.2.8	HAL_SAI_MspInit .....	705
48.2.9	HAL_SAI_MspDeInit .....	705
48.2.10	HAL_SAI_Transmit .....	705
48.2.11	HAL_SAI_Receive .....	705
48.2.12	HAL_SAI_Transmit_IT .....	705
48.2.13	HAL_SAI_Receive_IT .....	706
48.2.14	HAL_SAI_DMAPause .....	706
48.2.15	HAL_SAI_DMAResume .....	706
48.2.16	HAL_SAI_DMAStop .....	706
48.2.17	HAL_SAI_Abort .....	707
48.2.18	HAL_SAI_Transmit_DMA .....	707
48.2.19	HAL_SAI_Receive_DMA .....	707

48.2.20	HAL_SAI_EnableTxMuteMode .....	707
48.2.21	HAL_SAI_DisableTxMuteMode .....	707
48.2.22	HAL_SAI_EnableRxMuteMode .....	708
48.2.23	HAL_SAI_DisableRxMuteMode .....	708
48.2.24	HAL_SAI_IRQHandler .....	708
48.2.25	HAL_SAI_TxCpltCallback .....	708
48.2.26	HAL_SAI_TxHalfCpltCallback .....	708
48.2.27	HAL_SAI_RxCpltCallback .....	709
48.2.28	HAL_SAI_RxHalfCpltCallback .....	709
48.2.29	HAL_SAI_ErrorCallback .....	709
48.2.30	HAL_SAI_GetState .....	709
48.2.31	HAL_SAI_GetError .....	709
48.3	SAI Firmware driver defines .....	710
48.3.1	SAI .....	710
<b>49</b>	<b>HAL SAI Extension Driver .....</b>	<b>718</b>
49.1	SAIEx Firmware driver API description .....	718
49.1.1	SAI peripheral extension features .....	718
49.1.2	How to use this driver .....	718
49.1.3	Extension features Functions .....	718
49.1.4	SAI_BlockSynchroConfig .....	718
49.1.5	SAI_GetInputClock .....	718
<b>50</b>	<b>HAL SDRAM Generic Driver .....</b>	<b>719</b>
50.1	SDRAM Firmware driver registers structures .....	719
50.1.1	SDRAM_HandleTypeDef .....	719
50.2	SDRAM Firmware driver API description .....	719
50.2.1	How to use this driver .....	719
50.2.2	SDRAM Initialization and de_initialization functions .....	720
50.2.3	SDRAM Input and Output functions .....	720
50.2.4	SDRAM Control functions .....	720
50.2.5	SDRAM State functions .....	721
50.2.6	HAL_SDRAM_Init .....	721
50.2.7	HAL_SDRAM_DeInit .....	721
50.2.8	HAL_SDRAM_MspInit .....	721
50.2.9	HAL_SDRAM_MspDeInit .....	721
50.2.10	HAL_SDRAM_IRQHandler .....	722
50.2.11	HAL_SDRAM_RefreshErrorCallback .....	722
50.2.12	HAL_SDRAM_DMA_XferCpltCallback .....	722
50.2.13	HAL_SDRAM_DMA_XferErrorCallback .....	722

50.2.14	HAL_SDRAM_Read_8b .....	722
50.2.15	HAL_SDRAM_Write_8b .....	723
50.2.16	HAL_SDRAM_Read_16b .....	723
50.2.17	HAL_SDRAM_Write_16b .....	723
50.2.18	HAL_SDRAM_Read_32b .....	723
50.2.19	HAL_SDRAM_Write_32b .....	724
50.2.20	HAL_SDRAM_Read_DMA .....	724
50.2.21	HAL_SDRAM_Write_DMA .....	724
50.2.22	HAL_SDRAM_WriteProtection_Enable .....	725
50.2.23	HAL_SDRAM_WriteProtection_Disable .....	725
50.2.24	HAL_SDRAM_SendCommand .....	725
50.2.25	HAL_SDRAM_ProgramRefreshRate .....	725
50.2.26	HAL_SDRAM_SetAutoRefreshNumber .....	725
50.2.27	HAL_SDRAM_GetModeStatus .....	726
50.2.28	HAL_SDRAM_GetState .....	726
50.3	SDRAM Firmware driver defines .....	726
50.3.1	SDRAM .....	726
<b>51</b>	<b>HAL SD Generic Driver .....</b>	<b>727</b>
51.1	SD Firmware driver registers structures .....	727
51.1.1	SD_HandleTypeDef .....	727
51.1.2	HAL_SD_CSDTypeDef .....	728
51.1.3	HAL_SD_CIDTypeDef .....	730
51.1.4	HAL_SD_CardStatusTypeDef .....	731
51.1.5	HAL_SD_CardInfoTypeDef .....	731
51.2	SD Firmware driver API description .....	732
51.2.1	How to use this driver .....	732
51.2.2	Initialization and de-initialization functions .....	734
51.2.3	IO operation functions .....	734
51.2.4	Peripheral Control functions .....	734
51.2.5	Peripheral State functions .....	735
51.2.6	HAL_SD_Init .....	735
51.2.7	HAL_SD_DeInit .....	735
51.2.8	HAL_SD_MspInit .....	735
51.2.9	HAL_SD_MspDeInit .....	735
51.2.10	HAL_SD_ReadBlocks .....	736
51.2.11	HAL_SD_WriteBlocks .....	736
51.2.12	HAL_SD_ReadBlocks_DMA .....	736
51.2.13	HAL_SD_WriteBlocks_DMA .....	737

51.2.14	HAL_SD_CheckReadOperation .....	737
51.2.15	HAL_SD_CheckWriteOperation .....	737
51.2.16	HAL_SD_Erase .....	737
51.2.17	HAL_SD_IRQHandler .....	737
51.2.18	HAL_SD_XferCpltCallback .....	738
51.2.19	HAL_SD_XferErrorCallback .....	738
51.2.20	HAL_SD_DMA_RxCpltCallback .....	738
51.2.21	HAL_SD_DMA_RxErrorCallback .....	738
51.2.22	HAL_SD_DMA_TxCpltCallback .....	738
51.2.23	HAL_SD_DMA_TxErrorCallback .....	739
51.2.24	HAL_SD_Get_CardInfo .....	739
51.2.25	HAL_SD_WideBusOperation_Config .....	739
51.2.26	HAL_SD_StopTransfer .....	739
51.2.27	HAL_SD_HighSpeed .....	740
51.2.28	HAL_SD_SendSDStatus .....	740
51.2.29	HAL_SD_GetStatus .....	740
51.2.30	HAL_SD_GetCardStatus .....	740
51.3	SD Firmware driver defines .....	740
51.3.1	SD .....	740
<b>52</b>	<b>HAL SMARTCARD Generic Driver .....</b>	<b>753</b>
52.1	SMARTCARD Firmware driver registers structures .....	753
52.1.1	SMARTCARD_InitTypeDef .....	753
52.1.2	SMARTCARD_AdvFeatureInitTypeDef .....	754
52.1.3	SMARTCARD_HandleTypeDef .....	755
52.2	SMARTCARD Firmware driver API description .....	756
52.2.1	How to use this driver .....	756
52.2.2	Initialization and Configuration functions .....	757
52.2.3	IO operation functions .....	757
52.2.4	Peripheral State and Errors functions .....	758
52.2.5	HAL_SMARTCARD_Init .....	758
52.2.6	HAL_SMARTCARD_DeInit .....	758
52.2.7	HAL_SMARTCARD_MspInit .....	758
52.2.8	HAL_SMARTCARD_MspDeInit .....	758
52.2.9	HAL_SMARTCARD_Transmit .....	758
52.2.10	HAL_SMARTCARD_Receive .....	759
52.2.11	HAL_SMARTCARD_Transmit_IT .....	759
52.2.12	HAL_SMARTCARD_Receive_IT .....	759
52.2.13	HAL_SMARTCARD_Transmit_DMA .....	759

52.2.14	HAL_SMARTCARD_Receive_DMA.....	760
52.2.15	HAL_SMARTCARD_IRQHandler.....	760
52.2.16	HAL_SMARTCARD_TxCpltCallback .....	760
52.2.17	HAL_SMARTCARD_RxCpltCallback .....	760
52.2.18	HAL_SMARTCARD_ErrorCallback.....	760
52.2.19	HAL_SMARTCARD_GetState .....	761
52.2.20	HAL_SMARTCARD_GetError.....	761
52.3	SMARTCARD Firmware driver defines .....	761
52.3.1	SMARTCARD.....	761
<b>53</b>	<b>HAL SMARTCARD Extension Driver .....</b>	<b>771</b>
53.1	SMARTCARDEx Firmware driver API description .....	771
53.1.1	How to use this driver .....	771
53.1.2	Peripheral Control functions .....	771
53.1.3	HAL_SMARTCARDEx_BlockLength_Config .....	771
53.1.4	HAL_SMARTCARDEx_TimeOut_Config .....	771
53.1.5	HAL_SMARTCARDEx_EnableReceiverTimeOut .....	772
53.1.6	HAL_SMARTCARDEx_DisableReceiverTimeOut .....	772
<b>54</b>	<b>HAL SPDIFRX Generic Driver .....</b>	<b>773</b>
54.1	SPDIFRX Firmware driver registers structures .....	773
54.1.1	SPDIFRX_InitTypeDef.....	773
54.1.2	SPDIFRX_SetDataFormatTypeDef .....	774
54.1.3	SPDIFRX_HandleTypeDef .....	774
54.2	SPDIFRX Firmware driver API description.....	775
54.2.1	How to use this driver .....	775
54.2.2	Initialization and de-initialization functions .....	777
54.2.3	IO operation functions .....	777
54.2.4	Peripheral State and Errors functions .....	778
54.2.5	HAL_SPDIFRX_Init .....	778
54.2.6	HAL_SPDIFRX_DeInit.....	778
54.2.7	HAL_SPDIFRX_MspInit .....	778
54.2.8	HAL_SPDIFRX_MspDeInit.....	779
54.2.9	HAL_SPDIFRX_SetDataFormat .....	779
54.2.10	HAL_SPDIFRX_ReceiveDataFlow.....	779
54.2.11	HAL_SPDIFRX_ReceiveControlFlow.....	779
54.2.12	HAL_SPDIFRX_ReceiveDataFlow_IT .....	780
54.2.13	HAL_SPDIFRX_ReceiveControlFlow_IT .....	780
54.2.14	HAL_SPDIFRX_ReceiveDataFlow_DMA.....	780

54.2.15	HAL_SPDIFRX_ReceiveControlFlow_DMA.....	780
54.2.16	HAL_SPDIFRX_DMASop .....	780
54.2.17	HAL_SPDIFRX_IRQHandler .....	781
54.2.18	HAL_SPDIFRX_RxHalfCpltCallback.....	781
54.2.19	HAL_SPDIFRX_RxCpltCallback .....	781
54.2.20	HAL_SPDIFRX_CxHalfCpltCallback.....	781
54.2.21	HAL_SPDIFRX_CxCpltCallback .....	781
54.2.22	HAL_SPDIFRX_ErrorCallback .....	782
54.2.23	HAL_SPDIFRX_GetState .....	782
54.2.24	HAL_SPDIFRX_GetError .....	782
54.3	SPDIFRX Firmware driver defines .....	782
54.3.1	SPDIFRX .....	782
<b>55</b>	<b>HAL SPI Generic Driver .....</b>	<b>788</b>
55.1	SPI Firmware driver registers structures .....	788
55.1.1	SPI_InitTypeDef .....	788
55.1.2	__SPI_HandleTypeDef.....	789
55.2	SPI Firmware driver API description .....	790
55.2.1	How to use this driver .....	790
55.2.2	Initialization and de-initialization functions .....	790
55.2.3	IO operation functions .....	791
55.2.4	Peripheral State and Errors functions .....	792
55.2.5	HAL_SPI_Init.....	792
55.2.6	HAL_SPI_DeInit .....	792
55.2.7	HAL_SPI_MspInit .....	792
55.2.8	HAL_SPI_MspDeInit.....	792
55.2.9	HAL_SPI_Transmit.....	793
55.2.10	HAL_SPI_Receive.....	793
55.2.11	HAL_SPI_TransmitReceive.....	793
55.2.12	HAL_SPI_Transmit_IT.....	793
55.2.13	HAL_SPI_Receive_IT.....	794
55.2.14	HAL_SPI_TransmitReceive_IT .....	794
55.2.15	HAL_SPI_Transmit_DMA.....	794
55.2.16	HAL_SPI_Receive_DMA.....	794
55.2.17	HAL_SPI_TransmitReceive_DMA.....	795
55.2.18	HAL_SPI_DMAPause.....	795
55.2.19	HAL_SPI_DMAResume .....	795
55.2.20	HAL_SPI_DMASop .....	795
55.2.21	HAL_SPI_IRQHandler.....	796

55.2.22	HAL_SPI_TxCpltCallback .....	796
55.2.23	HAL_SPI_RxCpltCallback .....	796
55.2.24	HAL_SPI_TxRxCpltCallback .....	796
55.2.25	HAL_SPI_TxHalfCpltCallback .....	796
55.2.26	HAL_SPI_RxHalfCpltCallback .....	796
55.2.27	HAL_SPI_TxRxHalfCpltCallback .....	797
55.2.28	HAL_SPI_ErrorCallback .....	797
55.2.29	HAL_SPI_GetState .....	797
55.2.30	HAL_SPI_GetError .....	797
55.3	SPI Firmware driver defines .....	797
55.3.1	SPI .....	797
<b>56</b>	<b>HAL SRAM Generic Driver .....</b>	<b>805</b>
56.1	SRAM Firmware driver registers structures .....	805
56.1.1	SRAM_HandleTypeDef .....	805
56.2	SRAM Firmware driver API description .....	805
56.2.1	How to use this driver .....	805
56.2.2	SRAM Initialization and de_initialization functions .....	806
56.2.3	SRAM Input and Output functions .....	806
56.2.4	SRAM Control functions .....	806
56.2.5	SRAM State functions .....	807
56.2.6	HAL_SRAM_Init .....	807
56.2.7	HAL_SRAM_DeInit .....	807
56.2.8	HAL_SRAM_MspInit .....	807
56.2.9	HAL_SRAM_MspDeInit .....	807
56.2.10	HAL_SRAM_DMA_XferCpltCallback .....	808
56.2.11	HAL_SRAM_DMA_XferErrorCallback .....	808
56.2.12	HAL_SRAM_Read_8b .....	808
56.2.13	HAL_SRAM_Write_8b .....	808
56.2.14	HAL_SRAM_Read_16b .....	808
56.2.15	HAL_SRAM_Write_16b .....	809
56.2.16	HAL_SRAM_Read_32b .....	809
56.2.17	HAL_SRAM_Write_32b .....	809
56.2.18	HAL_SRAM_Read_DMA .....	810
56.2.19	HAL_SRAM_Write_DMA .....	810
56.2.20	HAL_SRAM_DMA_XferCpltCallback .....	810
56.2.21	HAL_SRAM_DMA_XferErrorCallback .....	810
56.2.22	HAL_SRAM_WriteOperation_Enable .....	810
56.2.23	HAL_SRAM_WriteOperation_Disable .....	811

---

56.2.24	HAL_SRAM_GetState .....	811
56.3	SRAM Firmware driver defines .....	811
56.3.1	SRAM .....	811
<b>57</b>	<b>HAL TIM Generic Driver .....</b>	<b>812</b>
57.1	TIM Firmware driver registers structures.....	812
57.1.1	TIM_Base_InitTypeDef.....	812
57.1.2	TIM_OC_InitTypeDef.....	812
57.1.3	TIM_OnePulse_InitTypeDef .....	813
57.1.4	TIM_IC_InitTypeDef .....	814
57.1.5	TIM_Encoder_InitTypeDef .....	814
57.1.6	TIM_ClockConfigTypeDef .....	815
57.1.7	TIM_ClearInputConfigTypeDef.....	816
57.1.8	TIM_SlaveConfigTypeDef .....	816
57.1.9	TIM_HandleTypeDef .....	817
57.2	TIM Firmware driver API description .....	817
57.2.1	TIMER Generic features.....	817
57.2.2	How to use this driver .....	818
57.2.3	Time Base functions .....	818
57.2.4	Time Output Compare functions .....	819
57.2.5	Time PWM functions .....	819
57.2.6	Time Input Capture functions .....	820
57.2.7	Time One Pulse functions .....	820
57.2.8	Time Encoder functions.....	821
57.2.9	IRQ handler management .....	821
57.2.10	Peripheral Control functions .....	821
57.2.11	TIM Callbacks functions .....	822
57.2.12	Peripheral State functions .....	822
57.2.13	HAL_TIM_Base_Init .....	822
57.2.14	HAL_TIM_Base_DeInit.....	823
57.2.15	HAL_TIM_Base_MspInit.....	823
57.2.16	HAL_TIM_Base_MspDeInit.....	823
57.2.17	HAL_TIM_Base_Start.....	823
57.2.18	HAL_TIM_Base_Stop.....	823
57.2.19	HAL_TIM_Base_Start_IT .....	823
57.2.20	HAL_TIM_Base_Stop_IT.....	824
57.2.21	HAL_TIM_Base_Start_DMA .....	824
57.2.22	HAL_TIM_Base_Stop_DMA.....	824
57.2.23	HAL_TIM_OC_Init .....	824



57.2.24	HAL_TIM_OC_DeInit.....	825
57.2.25	HAL_TIM_OC_MspInit .....	825
57.2.26	HAL_TIM_OC_MspDeInit.....	825
57.2.27	HAL_TIM_OC_Start .....	825
57.2.28	HAL_TIM_OC_Stop.....	825
57.2.29	HAL_TIM_OC_Start_IT .....	826
57.2.30	HAL_TIM_OC_Stop_IT .....	826
57.2.31	HAL_TIM_OC_Start_DMA .....	826
57.2.32	HAL_TIM_OC_Stop_DMA .....	827
57.2.33	HAL_TIM_PWM_Init.....	827
57.2.34	HAL_TIM_PWM_DeInit .....	827
57.2.35	HAL_TIM_PWM_MspInit.....	827
57.2.36	HAL_TIM_PWM_MspDeInit .....	827
57.2.37	HAL_TIM_PWM_Start.....	828
57.2.38	HAL_TIM_PWM_Stop .....	828
57.2.39	HAL_TIM_PWM_Start_IT.....	828
57.2.40	HAL_TIM_PWM_Stop_IT .....	829
57.2.41	HAL_TIM_PWM_Start_DMA.....	829
57.2.42	HAL_TIM_PWM_Stop_DMA .....	829
57.2.43	HAL_TIM_IC_Init.....	829
57.2.44	HAL_TIM_IC_DeInit .....	830
57.2.45	HAL_TIM_IC_MspInit .....	830
57.2.46	HAL_TIM_IC_MspDeInit.....	830
57.2.47	HAL_TIM_IC_Start .....	830
57.2.48	HAL_TIM_IC_Stop .....	830
57.2.49	HAL_TIM_IC_Start_IT.....	831
57.2.50	HAL_TIM_IC_Stop_IT .....	831
57.2.51	HAL_TIM_IC_Start_DMA .....	831
57.2.52	HAL_TIM_IC_Stop_DMA .....	832
57.2.53	HAL_TIM_OnePulse_Init.....	832
57.2.54	HAL_TIM_OnePulse_DeInit .....	832
57.2.55	HAL_TIM_OnePulse_MspInit.....	832
57.2.56	HAL_TIM_OnePulse_MspDeInit .....	833
57.2.57	HAL_TIM_OnePulse_Start.....	833
57.2.58	HAL_TIM_OnePulse_Stop .....	833
57.2.59	HAL_TIM_OnePulse_Start_IT.....	833
57.2.60	HAL_TIM_OnePulse_Stop_IT .....	834
57.2.61	HAL_TIM_Encoder_Init .....	834

57.2.62	HAL_TIM_Encoder_DelInit .....	834
57.2.63	HAL_TIM_Encoder_MspInit .....	834
57.2.64	HAL_TIM_Encoder_MspDelInit.....	835
57.2.65	HAL_TIM_Encoder_Start .....	835
57.2.66	HAL_TIM_Encoder_Stop .....	835
57.2.67	HAL_TIM_Encoder_Start_IT .....	835
57.2.68	HAL_TIM_Encoder_Stop_IT .....	836
57.2.69	HAL_TIM_Encoder_Start_DMA .....	836
57.2.70	HAL_TIM_Encoder_Stop_DMA .....	836
57.2.71	HAL_TIM_IRQHandler .....	837
57.2.72	HAL_TIM_OC_ConfigChannel .....	837
57.2.73	HAL_TIM_IC_ConfigChannel.....	837
57.2.74	HAL_TIM_PWM_ConfigChannel.....	837
57.2.75	HAL_TIM_OnePulse_ConfigChannel.....	838
57.2.76	HAL_TIM_DMABurst_WriteStart.....	838
57.2.77	HAL_TIM_DMABurst_WriteStop .....	839
57.2.78	HAL_TIM_DMABurst_ReadStart.....	839
57.2.79	HAL_TIM_DMABurst_ReadStop.....	840
57.2.80	HAL_TIM_GenerateEvent .....	840
57.2.81	HAL_TIM_ConfigOCrefClear.....	840
57.2.82	HAL_TIM_ConfigClockSource .....	841
57.2.83	HAL_TIM_ConfigTI1Input.....	841
57.2.84	HAL_TIM_SlaveConfigSynchronization .....	841
57.2.85	HAL_TIM_SlaveConfigSynchronization_IT .....	842
57.2.86	HAL_TIM_ReadCapturedValue.....	842
57.2.87	HAL_TIM_PeriodElapsedCallback .....	842
57.2.88	HAL_TIM_OC_DelayElapsedCallback.....	842
57.2.89	HAL_TIM_IC_CaptureCallback .....	843
57.2.90	HAL_TIM_PWM_PulseFinishedCallback .....	843
57.2.91	HAL_TIM_TriggerCallback .....	843
57.2.92	HAL_TIM_ErrorCallback.....	843
57.2.93	HAL_TIM_Base_GetState .....	843
57.2.94	HAL_TIM_OC_GetState.....	844
57.2.95	HAL_TIM_PWM_GetState .....	844
57.2.96	HAL_TIM_IC_GetState.....	844
57.2.97	HAL_TIM_OnePulse_GetState .....	844
57.2.98	HAL_TIM_Encoder_GetState.....	844
57.3	TIM Firmware driver defines.....	845
57.3.1	TIM.....	845

<b>58</b>	<b>HAL TIM Extension Driver.....</b>	<b>858</b>
58.1	TIMEx Firmware driver registers structures.....	858
58.1.1	TIM_HallSensor_InitTypeDef .....	858
58.1.2	TIM_MasterConfigTypeDef .....	858
58.1.3	TIM_BreakDeadTimeConfigTypeDef .....	859
58.2	TIMEx Firmware driver API description.....	860
58.2.1	TIMER Extended features .....	860
58.2.2	How to use this driver .....	860
58.2.3	Timer Hall Sensor functions .....	861
58.2.4	Timer Complementary Output Compare functions.....	861
58.2.5	Timer Complementary PWM functions.....	861
58.2.6	Timer Complementary One Pulse functions.....	862
58.2.7	Peripheral Control functions .....	862
58.2.8	Extension Callbacks functions.....	863
58.2.9	Extension Peripheral State functions .....	863
58.2.10	HAL_TIMEx_HallSensor_Init.....	863
58.2.11	HAL_TIMEx_HallSensor_DeInit.....	863
58.2.12	HAL_TIMEx_HallSensor_MspInit.....	863
58.2.13	HAL_TIMEx_HallSensor_MspDeInit .....	863
58.2.14	HAL_TIMEx_HallSensor_Start.....	864
58.2.15	HAL_TIMEx_HallSensor_Stop .....	864
58.2.16	HAL_TIMEx_HallSensor_Start_IT.....	864
58.2.17	HAL_TIMEx_HallSensor_Stop_IT .....	864
58.2.18	HAL_TIMEx_HallSensor_Start_DMA.....	864
58.2.19	HAL_TIMEx_HallSensor_Stop_DMA .....	865
58.2.20	HAL_TIMEx_OCN_Start.....	865
58.2.21	HAL_TIMEx_OCN_Stop.....	865
58.2.22	HAL_TIMEx_OCN_Start_IT .....	866
58.2.23	HAL_TIMEx_OCN_Stop_IT .....	866
58.2.24	HAL_TIMEx_OCN_Start_DMA .....	866
58.2.25	HAL_TIMEx_OCN_Stop_DMA.....	866
58.2.26	HAL_TIMEx_PWMN_Start .....	867
58.2.27	HAL_TIMEx_PWMN_Stop .....	867
58.2.28	HAL_TIMEx_PWMN_Start_IT .....	867
58.2.29	HAL_TIMEx_PWMN_Stop_IT .....	868
58.2.30	HAL_TIMEx_PWMN_Start_DMA .....	868
58.2.31	HAL_TIMEx_PWMN_Stop_DMA .....	868
58.2.32	HAL_TIMEx_OnePulseN_Start .....	869

58.2.33	HAL_TIMEx_OnePulseN_Stop .....	869
58.2.34	HAL_TIMEx_OnePulseN_Start_IT .....	869
58.2.35	HAL_TIMEx_OnePulseN_Stop_IT .....	869
58.2.36	HAL_TIMEx_ConfigCommutationEvent .....	870
58.2.37	HAL_TIMEx_ConfigCommutationEvent_IT .....	870
58.2.38	HAL_TIMEx_ConfigCommutationEvent_DMA .....	871
58.2.39	HAL_TIM_OC_ConfigChannel .....	872
58.2.40	HAL_TIM_PWM_ConfigChannel .....	872
58.2.41	HAL_TIM_ConfigOCrefClear .....	872
58.2.42	HAL_TIMEx_MasterConfigSynchronization .....	873
58.2.43	HAL_TIMEx_ConfigBreakDeadTime .....	873
58.2.44	HAL_TIMEx_RemapConfig .....	873
58.2.45	HAL_TIMEx_GroupChannel5 .....	874
58.2.46	HAL_TIMEx_CommutationCallback .....	874
58.2.47	HAL_TIMEx_BreakCallback .....	874
58.2.48	HAL_TIMEx_DMACommutationCplt .....	875
58.2.49	HAL_TIMEx_HallSensor_GetState .....	875
58.3	TIMEx Firmware driver defines .....	875
58.3.1	TIMEx .....	875
<b>59</b>	<b>HAL UART Generic Driver .....</b>	<b>879</b>
59.1	UART Firmware driver registers structures .....	879
59.1.1	UART_InitTypeDef .....	879
59.1.2	UART_AdvFeatureInitTypeDef .....	880
59.1.3	UART_HandleTypeDef .....	880
59.2	UART Firmware driver API description .....	882
59.2.1	How to use this driver .....	882
59.2.2	Initialization and Configuration functions .....	884
59.2.3	IO operation functions .....	884
59.2.4	Peripheral Control functions .....	884
59.2.5	HAL_UART_Init .....	885
59.2.6	HAL_HalfDuplex_Init .....	885
59.2.7	HAL_LIN_Init .....	885
59.2.8	HAL_MultiProcessor_Init .....	886
59.2.9	HAL_UART_DeInit .....	886
59.2.10	HAL_UART_MspInit .....	886
59.2.11	HAL_UART_MspDeInit .....	887
59.2.12	HAL_UART_Transmit .....	887
59.2.13	HAL_UART_Receive .....	887

59.2.14	HAL_UART_Transmit_IT.....	887
59.2.15	HAL_UART_Receive_IT.....	887
59.2.16	HAL_UART_Transmit_DMA.....	888
59.2.17	HAL_UART_Receive_DMA.....	888
59.2.18	HAL_UART_DMABase.....	888
59.2.19	HAL_UART_DMAResume .....	888
59.2.20	HAL_UART_DMABase .....	888
59.2.21	HAL_UART_IRQHandler.....	889
59.2.22	UART_WaitOnFlagUntilTimeout .....	889
59.2.23	HAL_UART_TxCpltCallback .....	889
59.2.24	HAL_UART_TxHalfCpltCallback.....	889
59.2.25	HAL_UART_RxCpltCallback .....	889
59.2.26	HAL_UART_RxHalfCpltCallback.....	890
59.2.27	HAL_UART_ErrorCallback .....	890
59.2.28	HAL_MultiProcessor_EnableMuteMode .....	890
59.2.29	HAL_MultiProcessor_DisableMuteMode.....	890
59.2.30	HAL_MultiProcessor_EnterMuteMode .....	890
59.2.31	HAL_UART_GetState.....	891
59.2.32	HAL_UART_GetError .....	891
59.2.33	UART_SetConfig .....	891
59.2.34	UART_AdvFeatureConfig.....	891
59.2.35	UART_CheckIdleState .....	891
59.2.36	HAL_HalfDuplex_EnableTransmitter .....	891
59.2.37	HAL_HalfDuplex_EnableReceiver .....	892
59.2.38	HAL_LIN_SendBreak .....	892
59.2.39	HAL_UART_GetState.....	892
59.2.40	HAL_UART_GetError .....	892
59.3	UART Firmware driver defines .....	892
59.3.1	UART .....	892
<b>60</b>	<b>HAL UART Extension Driver .....</b>	<b>909</b>
60.1	UARTEEx Firmware driver defines.....	909
60.1.1	UARTEEx.....	909
<b>61</b>	<b>HAL USART Generic Driver .....</b>	<b>910</b>
61.1	USART Firmware driver registers structures.....	910
61.1.1	USART_InitTypeDef .....	910
61.1.2	USART_HandleTypeDef .....	911
61.2	USART Firmware driver API description .....	912

61.2.1	How to use this driver .....	912
61.2.2	Initialization and Configuration functions .....	912
61.2.3	IO operation functions .....	913
61.2.4	Peripheral State and Errors functions .....	914
61.2.5	HAL_USART_Init .....	914
61.2.6	HAL_USART_DeInit .....	915
61.2.7	HAL_USART_MspInit .....	915
61.2.8	HAL_USART_MspDeInit .....	915
61.2.9	HAL_USART_CheckIdleState .....	915
61.2.10	HAL_USART_Transmit .....	915
61.2.11	HAL_USART_Receive .....	915
61.2.12	HAL_USART_TransmitReceive .....	916
61.2.13	HAL_USART_Transmit_IT .....	916
61.2.14	HAL_USART_Receive_IT .....	916
61.2.15	HAL_USART_TransmitReceive_IT .....	916
61.2.16	HAL_USART_Transmit_DMA .....	917
61.2.17	HAL_USART_Receive_DMA .....	917
61.2.18	HAL_USART_TransmitReceive_DMA .....	917
61.2.19	HAL_USART_DMAPause .....	918
61.2.20	HAL_USART_DMAResume .....	918
61.2.21	HAL_USART_DMAStop .....	918
61.2.22	HAL_USART_IRQHandler .....	918
61.2.23	HAL_USART_TxCpltCallback .....	918
61.2.24	HAL_USART_TxHalfCpltCallback .....	919
61.2.25	HAL_USART_RxCpltCallback .....	919
61.2.26	HAL_USART_RxHalfCpltCallback .....	919
61.2.27	HAL_USART_TxRxCpltCallback .....	919
61.2.28	HAL_USART_ErrorCallback .....	919
61.2.29	HAL_USART_GetState .....	919
61.2.30	HAL_USART_GetError .....	920
61.3	USART Firmware driver defines .....	920
61.3.1	USART .....	920
<b>62</b>	<b>HAL USART Extension Driver .....</b>	<b>927</b>
62.1	USARTEx Firmware driver defines .....	927
62.1.1	USARTEx .....	927
<b>63</b>	<b>HAL WWDG Generic Driver .....</b>	<b>928</b>
63.1	WWDG Firmware driver registers structures .....	928
63.1.1	WWDG_InitTypeDef .....	928

63.1.2	WWDG_HandleTypeDef .....	928
63.2	WWDG Firmware driver API description .....	929
63.2.1	WWDG specific features .....	929
63.2.2	How to use this driver .....	929
63.2.3	Initialization and de-initialization functions .....	929
63.2.4	IO operation functions .....	930
63.2.5	Peripheral State functions .....	930
63.2.6	HAL_WWDG_Init.....	930
63.2.7	HAL_WWDG_DeInit .....	930
63.2.8	HAL_WWDG_MspInit.....	931
63.2.9	HAL_WWDG_MspDeInit .....	931
63.2.10	HAL_WWDG_WakeupCallback .....	931
63.2.11	HAL_WWDG_Start.....	931
63.2.12	HAL_WWDG_Start_IT.....	931
63.2.13	HAL_WWDG_Refresh.....	932
63.2.14	HAL_WWDG_IRQHandler .....	932
63.2.15	HAL_WWDG_WakeupCallback .....	932
63.2.16	HAL_WWDG_GetState .....	932
63.3	WWDG Firmware driver defines.....	933
63.3.1	WWDG.....	933
64	FAQs.....	937
65	Revision history .....	941

## List of tables

Table 1: Acronyms and definitions.....	54
Table 2: HAL drivers files.....	56
Table 3: User-application files .....	57
Table 4: APis classification .....	62
Table 5: List of devices supported by HAL drivers .....	63
Table 6: HAL API naming rules .....	66
Table 7: Macros handling interrupts and specific clock configurations .....	67
Table 8: Callback functions.....	68
Table 9: HAL generic APIs .....	69
Table 10: HAL extension APIs.....	70
Table 11: Define statements used for HAL configuration .....	73
Table 12: Description of GPIO_InitTypeDef structure .....	75
Table 13: Description of EXTI configuration macros .....	77
Table 14: MSP functions.....	82
Table 15: Timeout values .....	86
Table 16: USART frame formats .....	757
Table 17: USART frame formats .....	913
Table 18: Document revision history .....	941



## List of figures

Figure 1: Example of project template .....	59
Figure 2: Adding device-specific functions .....	70
Figure 3: Adding family-specific functions .....	71
Figure 4: Adding new peripherals .....	71
Figure 5: Updating existing APIs .....	71
Figure 6: File inclusion model .....	72
Figure 7: HAL driver model .....	80

# 1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
CAN	Controller area network
CEC	Consumer electronic controller
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
ETH	Ethernet controller
EXTI	External interrupt/event controller
FLASH	Flash memory
FMC	Flexible memory controller
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
HCD	USB Host Controller Driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controller
LTDC	LCD TFT Display Controller
MSP	MCU Specific Package
NAND	NAND Flash memory
NOR	Nor Flash memory
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
QSPI	QUADSPI Flash memory Interface
RCC	Reset and clock controller
RTC	Real-time clock
SAI	Serial Audio Interface

Acronym	Definition
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SPDIFRX	SPDIF-RX Receiver 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: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode.
- Peripheral multi-instance support allowing concurrent API calls for multiple instances of a given peripheral (USART1, USART2...)
- All HAL APIs implement user-callback functions mechanism:
  - Peripheral Init/Delinit 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 timbase.

### 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>stm32f7xx_hal_ppp.c</i>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32f7xx_hal_adc.c, stm32f7xx_hal_irda.c, ...</i>
<i>stm32f7xxx_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: stm32f7xxx_hal_adc.h, stm32f7xxx_hal_irda.h, ...</i>

File	Description
<i>stm32f7xxx_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:</i> <i>stm32f7xxx_hal_adc_ex.c, stm32f7xxx_hal_dma_ex.c, ...</i>
<i>stm32f7xxx_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: stm32f7xx</i> <i>xx_hal_adc_ex.h, stm32f7xxx_hal_dma_ex.h, ...</i>
<i>stm32f7xxx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32f7xxx_hal.h</i>	<i>stm32f7xxx_hal.c</i> header file
<i>stm32f7xxx_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>stm32f7xxx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32f7xxx_hal_def.h</i>	Common HAL resources such as common define statements, enumerations, structures and macros.

### 2.1.2 User-application files

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

**Table 3: User-application files**

File	Description
<i>system_stm32f7xx.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> </ul>
<i>startup_stm32f7xx.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>stm32f7xx_flash.icf</i> (optional)	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32f7xx_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>stm32f7xx_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.

File	Description
<i>stm32f7xx_it.c/h</i>	<p>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>stm32f7xx_hal.c</i>) used as HAL timebase. By default, this function is called each 1ms in SysTick ISR. .</p> <p>The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.</p>
<i>main.c/h</i>	<p>This file contains the main program routine, mainly:</p> <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.

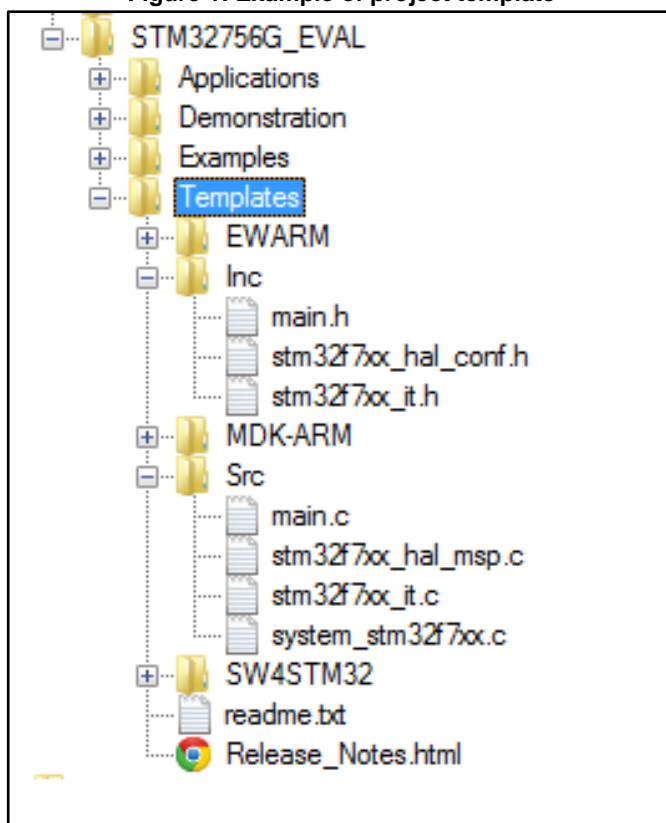
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- 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 HAL_USART_ErrorTypeDef ErrorCode; /* USART Error code */
}USART_HandleTypeDef;
```



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

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



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



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

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

## 2.2.2 Initialization and configuration structure

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

```
typedef struct
{
  uint32_t BaudRate; /*!< This member configures the UART communication baudrate.*/
  uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received
```



```

in a frame.*/
uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
uint32_t Parity; /*!< Specifies the parity mode. */
uint32_t Mode; /*!< Specifies 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_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t
SingleDiff); uint32_t HAL_ADCEX_Calibration_GetValue(ADC_HandleTypeDef* hadc,
uint32_t SingleDiff);

```

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

```

#if defined(STM32F756xx)
HAL_StatusTypeDef HAL_HASHEx_SHA224_Accumulate(HASH_HandleTypeDef* hhash, uint8_t
*pInBuffer, uint32_t Size);
#endif /* STM32F756xx */

```



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

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

**Table 4: APIs classification**

	Generic file	Extension file
<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



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



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

## 2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

IP/Module	STM32F745xx	STM32F746xx	STM32F756xx
stm32f7xx_hal.c	Yes	Yes	Yes
stm32f7xx_hal_adc.c	Yes	Yes	Yes
stm32f7xx_hal_adc_ex.c	Yes	Yes	Yes
stm32f7xx_hal_can.c	Yes	Yes	Yes
stm32f7xx_hal_cec.c	Yes	Yes	Yes
stm32f7xx_hal_cortex.c	Yes	Yes	Yes
stm32f7xx_hal_crc.c	Yes	Yes	Yes
stm32f7xx_hal_crc_ex.c	Yes	Yes	Yes
stm32f7xx_hal_cryp.c	No	No	Yes
stm32f7xx_hal_cryp_ex.c	No	No	Yes
stm32f7xx_hal_dac.c	Yes	Yes	Yes
stm32f7xx_hal_dac_ex.c	Yes	Yes	Yes
stm32f7xx_hal_dcmi.c	Yes	Yes	Yes
stm32f7xx_hal_dcmi_ex.c	Yes	Yes	Yes
stm32f7xx_hal_dma.c	Yes	Yes	Yes
stm32f7xx_hal_dma2d.c	Yes	Yes	Yes
stm32f7xx_hal_dma_ex.c	Yes	Yes	Yes
stm32f7xx_hal_eth.c	Yes	Yes	Yes
stm32f7xx_hal_flash.c	Yes	Yes	Yes
stm32f7xx_hal_flash_ex.c	Yes	Yes	Yes
stm32f7xx_hal_gpio.c	Yes	Yes	Yes
stm32f7xx_hal_hash.c	No	No	Yes
stm32f7xx_hal_hash_ex.c	No	No	Yes

## Overview of HAL drivers

UM1905

IP/Module	STM32F745xx	STM32F746xx	STM32F756xx
stm32f7xx_hal_hcd.c	Yes	Yes	Yes
stm32f7xx_hal_i2c.c	Yes	Yes	Yes
stm32f7xx_hal_i2c_ex.c	Yes	Yes	Yes
stm32f7xx_hal_i2s.c	Yes	Yes	Yes
stm32f7xx_hal_irda.c	Yes	Yes	Yes
stm32f7xx_hal_iwdg.c	Yes	Yes	Yes
stm32f7xx_hal_lptim.c	Yes	Yes	Yes
stm32f7xx_hal_ltdc.c	No	Yes	Yes
stm32f7xx_hal_msp_template.c	Yes	Yes	Yes
stm32f7xx_hal_nand.c	Yes	Yes	Yes
stm32f7xx_hal_nor.c	Yes	Yes	Yes
stm32f7xx_hal_pcd.c	Yes	Yes	Yes
stm32f7xx_hal_pcd_ex.c	Yes	Yes	Yes
stm32f7xx_hal_pwr.c	Yes	Yes	Yes
stm32f7xx_hal_pwr_ex.c	Yes	Yes	Yes
stm32f7xx_hal_qspi.c	Yes	Yes	Yes
stm32f7xx_hal_rcc.c	Yes	Yes	Yes
stm32f7xx_hal_rcc_ex.c	Yes	Yes	Yes
stm32f7xx_hal_rng.c	Yes	Yes	Yes
stm32f7xx_hal_rtc.c	Yes	Yes	Yes
stm32f7xx_hal_rtc_ex.c	Yes	Yes	Yes
stm32f7xx_hal_sai.c	Yes	Yes	Yes
stm32f7xx_hal_sai_ex.c	Yes	Yes	Yes
stm32f7xx_hal_sd.c	Yes	Yes	Yes

IP/Module	STM32F745xx	STM32F746xx	STM32F756xx
stm32f7xx_hal_sdram.c	Yes	Yes	Yes
stm32f7xx_hal_smartcard.c	Yes	Yes	Yes
stm32f7xx_hal_smartcard_ex.c	Yes	Yes	Yes
stm32f7xx_hal_spdifrx.c	Yes	Yes	Yes
stm32f7xx_hal_spi.c	Yes	Yes	Yes
stm32f7xx_hal_sram.c	Yes	Yes	Yes
stm32f7xx_hal_tim.c	Yes	Yes	Yes
stm32f7xx_hal_tim_ex.c	Yes	Yes	Yes
stm32f7xx_hal_uart.c	Yes	Yes	Yes
stm32f7xx_hal_usart.c	Yes	Yes	Yes
stm32f7xx_hal_wwdg.c	Yes	Yes	Yes
stm32f7xx_ll_fmc.c	Yes	Yes	Yes
stm32f7xx_ll_sdmmc.c	Yes	Yes	Yes
stm32f7xx_ll_usb.c	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
<b>File names</b>	<i>stm32f7xx_hal_ppp (c/h)</i>	<i>stm32f7xx_hal_ppp_ex (c/h)</i>	<i>stm32f7xx_hal_ppp_ex (c/h)</i>
<b>Module name</b>	<i>HAL_PPP_MODULE</i>		
<b>Function name</b>	<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>
<b>Handle name</b>	<i>PPP_HandleTypeDef</i>	NA	NA
<b>Init structure name</b>	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
<b>Enum name</b>	<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 stm32f7xx reference manuals.
- Peripheral registers are declared in the *PPP\_TypeDef* structure (e.g. *ADC\_TypeDef*) in *stm32f7xxx.h* header file. *stm32f7xxx.h* corresponds to *stm32f756xx.h*, *stm32f746xx.h* and *stm32f745xx.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 *stm32f7xx\_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 stm32f7xx\_it.c
- User callback functions.

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

There are three types of user callbacks functions:

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

**Table 8: Callback functions**

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

## 2.6 HAL generic APIs

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

- Initialization and de-initialization functions:** HAL\_PPP\_Init(), HAL\_PPP\_DeInit()
- IO operation functions:** HAL\_PPP\_Read(), HAL\_PPP\_Write(), HAL\_PPP\_Transmit(), HAL\_PPP\_Receive()
- Control functions:** HAL\_PPP\_Set (), HAL\_PPP\_Get ().
- State and Errors functions:** HAL\_PPP\_GetState (), HAL\_PPP\_GetError ().

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

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



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

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

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

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

**Table 9: HAL generic APIs**

Function Group	Common API Name	Description
Initialization group	<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.
IO operation group	<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>HAL_ADC_ErrorCallback()</i>	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	<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
State and Errors group	<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, `stm32f7xx_hal_ppp_ex.c`, that includes all the specific functions and define statements (`stm32f7xx_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
<i>HAL_ADCEx_CalibrationStart()</i>	This function is used to start the automatic ADC calibration

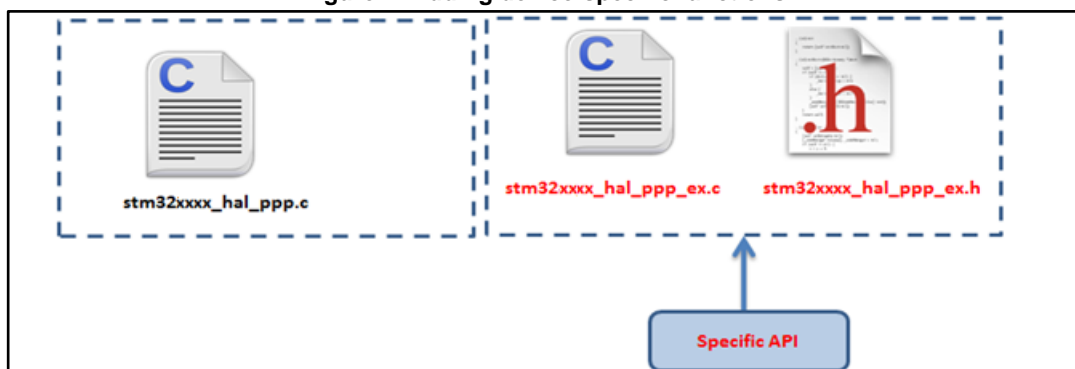
### 2.7.2 HAL extension model cases

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

#### Case1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the `stm32f7xx_hal_ppp_ex.c` extension file. They are named `HAL_PPPEX_Function()`.

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



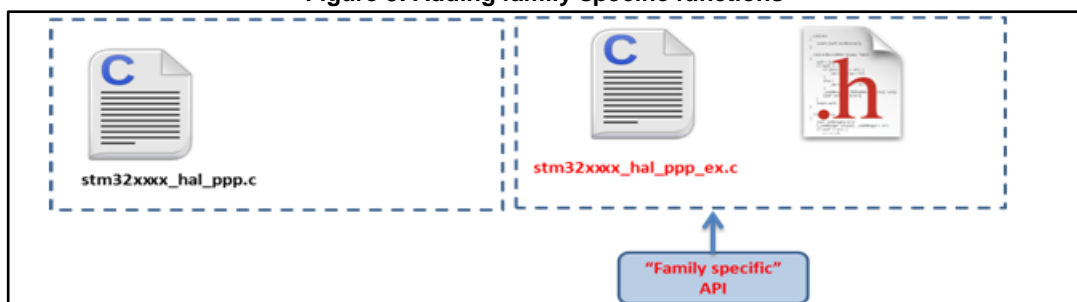
Example: `stm32f7xx_hal_hash_ex.h`

```
#if defined(STM32F756xx)
HAL StatusTypeDef HAL HASHEx SHA224 Accumulate(HASH HandleTypeDef *hhash, uint8 t
*pInBuffer, uint32 t Size);
HAL StatusTypeDef HAL HASHEx SHA256 Accumulate(HASH HandleTypeDef *hhash, uint8 t
*pInBuffer, uint32 t Size);
#endif /* STM32F756xx */
```

#### Case2: Adding a family-specific function

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

Figure 3: Adding family-specific functions

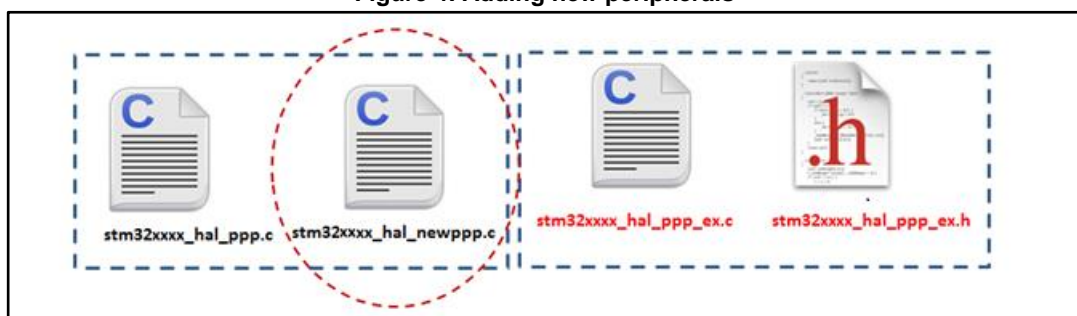


### Case3 : Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32f7xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32f7_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

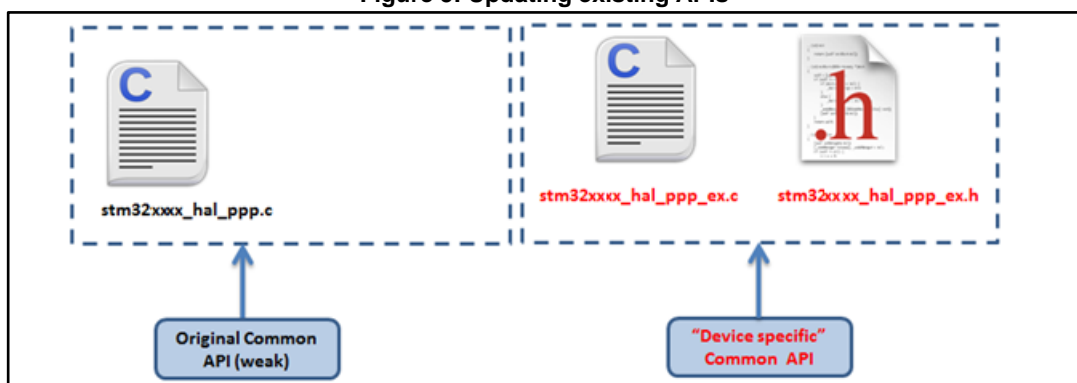


Example: `xx_hal_sai.c/h`

### Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the `stm32f7xx_hal_ppp_ex.c` extension file, while the generic API is defined as *weak*, so that the compiler will overwrite the original routine by the new defined function.

Figure 5: Updating existing APIs



### Case5 : Updating existing data structures

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

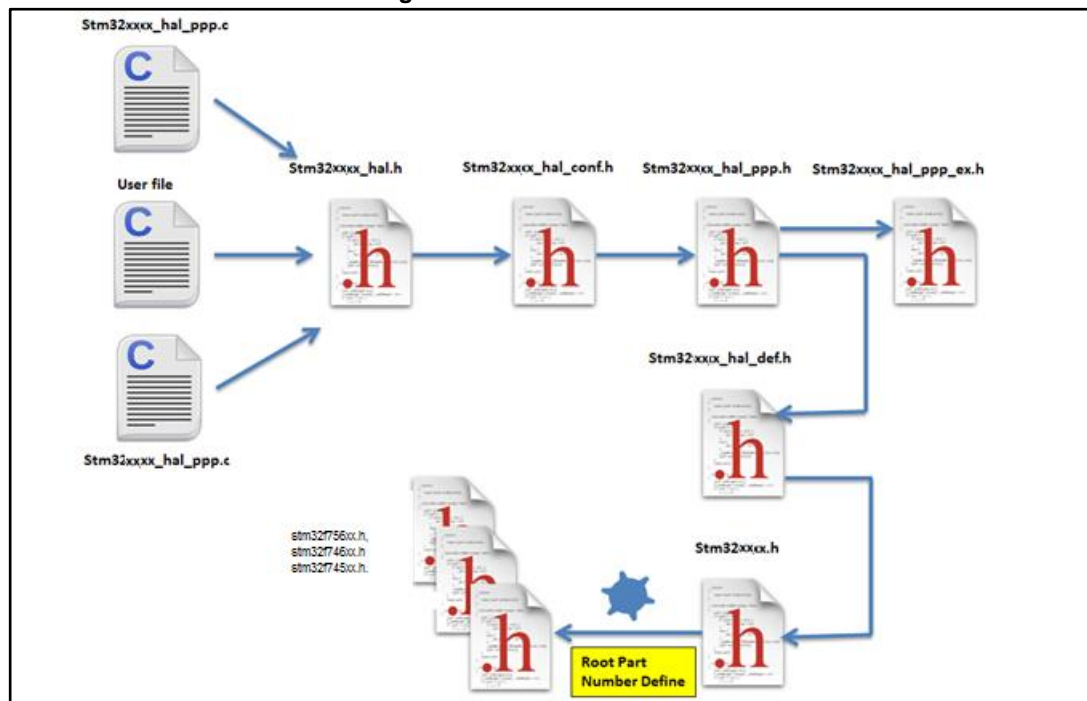
Example:

```
#if defined (STM32F756xx)
typedef struct
{
    (...)
} PPP_InitTypeDef;
#endif /* STM32F756xx */
```

## 2.8 File inclusion model

The header of the common HAL driver file (stm32f7xx\_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 6: File inclusion model



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

```
/* *****
 * @file stm32f7xx_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 *stm32f7xx\_hal\_def.h*. The main common define enumeration is *HAL\_StatusTypeDef*.

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

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

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

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

- In addition to common resources, the *stm32f7xx\_hal\_def.h* file calls the *stm32f7xx.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, *stm32f7xx\_hal\_conf.h*, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

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

**Table 11: Define statements used for HAL configuration**

Configuration item	Description	Default Value
<b>HSE_VALUE</b>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 Hz
<b>HSE_STARTUP_TIMEOUT</b>	Timeout for HSE start up, expressed in ms	5000
<b>HSI_VALUE</b>	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 Hz

Configuration item	Description	Default Value
<b>LSE_VALUE</b>	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz
<b>LSE_STARTUP_TIMEOUT</b>	Timeout for LSE start up, expressed in ms	5000
<b>VDD_VALUE</b>	VDD value	3300 (mV)
<b>USE_RTOS</b>	Enables the use of RTOS	FALSE (for future use)
<b>PREFETCH_ENABLE</b>	Enables prefetch feature	TRUE



The `stm32f7xx_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 `stm32f7xx_hal_conf_template.h` file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

## 2.11 HAL system peripheral handling

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

### 2.11.1 Clock

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

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

Some peripheral clocks are not derived from the system clock (RTC, USB...). In this case, the clock configuration is performed by an extended API defined in `stm32f7xx_hal_rcc_ex.c`: `HAL_RCCEx_PeriphCLKConfig` (`RCC_PeriphCLKInitTypeDef *PeriphClkInit`).

Additional RCC HAL driver functions are available:

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

A set of macros are defined in `stm32f7xx_hal_rcc.h` and `stm32f7xx_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 `stm32f7xx_it.c` and implement `HAL_GPIO_EXTI_Callback()`

The table below describes the `GPIO_InitTypeDef` structure field.

**Table 12: Description of `GPIO_InitTypeDef` structure**

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: <code>GPIO_PIN_x</code> or <code>GPIO_PIN_All</code> , where <code>x[0..15]</code>
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>– <code>GPIO_MODE_INPUT</code> : Input Floating</li> <li>– <code>GPIO_MODE_OUTPUT_PP</code> : Output Push Pull</li> <li>– <code>GPIO_MODE_OUTPUT_OD</code> : Output Open Drain</li> <li>– <code>GPIO_MODE_AF_PP</code> : Alternate Function Push Pull</li> <li>– <code>GPIO_MODE_AF_OD</code> : Alternate Function Open Drain</li> <li>– <code>GPIO_MODE_ANALOG</code> : Analog mode</li> </ul> </li> <li>• <u>External Interrupt Mode</u> <ul style="list-style-type: none"> <li>– <code>GPIO_MODE_IT_RISING</code> : Rising edge trigger detection</li> <li>– <code>GPIO_MODE_IT_FALLING</code> : Falling edge trigger detection</li> <li>– <code>GPIO_MODE_IT_RISING_FALLING</code> : Rising/Falling edge trigger detection</li> </ul> </li> <li>• <u>External Event Mode</u> <ul style="list-style-type: none"> <li>– <code>GPIO_MODE_EVT_RISING</code> : Rising edge trigger detection</li> <li>– <code>GPIO_MODE_EVT_FALLING</code> : Falling edge trigger detection</li> <li>– <code>GPIO_MODE_EVT_RISING_FALLING</code> : Rising/Falling edge trigger detection</li> </ul> </li> </ul>

Structure field	Description
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_LOW GPIO_SPEED_MEDIUM GPIO_SPEED_HIGH

Please find below typical GPIO configuration examples:

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

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

- Configuring PA0 as external interrupt with falling edge sensitivity:

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

### 2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, stm32f7xx\_hal\_cortex.c, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL\_NVIC\_SetPriority()/ HAL\_NVIC\_SetPriorityGrouping()
- HAL\_NVIC\_GetPriority() / HAL\_NVIC\_GetPriorityGrouping()
- HAL\_NVIC\_EnableIRQ()/HAL\_NVIC\_DisableIRQ()
- HAL\_NVIC\_SystemReset()
- HAL\_SYSTICK\_IRQHandler()
- HAL\_NVIC\_GetPendingIRQ() / HAL\_NVIC\_SetPendingIRQ () / HAL\_NVIC\_ClearPendingIRQ()
- HAL\_NVIC\_GetActive(IRQn)
- HAL\_SYSTICK\_Config()
- HAL\_SYSTICK\_CLKSourceConfig()
- HAL\_SYSTICK\_Callback()

### 2.11.4 PWR

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

- PVD configuration, enabling/disabling and interrupt handling
  - HAL\_PWR\_ConfigPVD()
  - HAL\_PWR\_EnablePVD() / HAL\_PWR\_DisablePVD()
  - HAL\_PWR\_PVD\_IRQHandler()
  - HAL\_PWR\_PVDCallback()
- Wakeup pin configuration
  - HAL\_PWR\_EnableWakeUpPin() / HAL\_PWR\_DisableWakeUpPin()



- Low power mode entry
  - HAL\_PWR\_EnterSLEEPMode()
  - HAL\_PWR\_EnterSTOPMode()
  - HAL\_PWR\_EnterSTANDBYMode()

### 2.11.5 EXTI

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

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

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

**Table 13: Description of EXTI configuration macros**

Macros	Description
<code>__HAL_PPP_{SUBBLOCK}_EXTI_ENABLE_IT()</code>	Enables a given EXTI line interrupt Example: <code>__HAL_PWR_PVD_EXTI_ENABLE_IT()</code>
<code>__HAL_PPP_{SUBBLOCK}_EXTI_DISABLE_IT()</code>	Disables a given EXTI line. Example: <code>__HAL_PWR_PVD_EXTI_DISABLE_IT()</code>
<code>__HAL_PPP_{SUBBLOCK}_EXTI_GET_FLAG()</code>	Gets a given EXTI line interrupt flag pending bit status. Example: <code>__HAL_PWR_PVD_EXTI_GET_FLAG()</code>
<code>__HAL_PPP_{SUBBLOCK}_EXTI_CLEAR_FLAG()</code>	Clears a given EXTI line interrupt flag pending bit. Example; <code>__HAL_PWR_PVD_EXTI_CLEAR_FLAG()</code>
<code>__HAL_PPP_{SUBBLOCK}_EXTI_GENERATE_SWIT()</code>	Generates a software interrupt for a given EXTI line. Example: <code>__HAL_PWR_PVD_EXTI_GENERATE_SWIT()</code>
<code>__HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()</code>	Enable a given EXTI line event Example: <code>__HAL_RTC_WAKEUP_EXTI_ENABLE_EVENT()</code>
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()</code>	Disable a given EXTI line event Example: <code>__HAL_RTC_WAKEUP_EXTI_DISABLE_EVENT()</code>

Macros	Description
<code>__HAL_PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()</code>	Configure an EXTI Interrupt or Event on rising edge
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()</code>	Enable an EXTI Interrupt or Event on Falling edge
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()</code>	Disable an EXTI Interrupt or Event on rising edge
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()</code>	Disable an EXTI Interrupt or Event on Falling edge
<code>__HAL_PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()</code>	Enable an EXTI Interrupt or Event on Rising/Falling edge
<code>__HAL_PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING_EDGE()</code>	Disable an EXTI Interrupt or Event on Rising/Falling edge

If the EXTI interrupt mode is selected, the user application must call `HAL_PPP_FUNCTION_IRQHandler()` (for example `HAL_PWR_PVD_IRQHandler()`), from `stm32f7xx_it.c` file, and implement `HAL_PPP_FUNCTIONCallback()` callback function (for example `HAL_PWR_PVDCallback()`).

### 2.11.6 DMA

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

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

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

Two operating modes are available:

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

- c. Use `HAL_DMA_Start_IT()` to start DMA transfer when the source and destination addresses and the length of data to be transferred have been 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`: enable the specified DMA Channels.
- `__HAL_DMA_DISABLE`: disables the specified DMA Channels.
- `__HAL_DMA_GET_FLAG`: gets the DMA Channels pending flags.
- `__HAL_DMA_CLEAR_FLAG`: clears the DMA Channels pending flags.
- `__HAL_DMA_ENABLE_IT`: enables the specified DMA Channels interrupts.
- `__HAL_DMA_DISABLE_IT`: disables the specified DMA Channels interrupts.
- `__HAL_DMA_GET_IT_SOURCE`: checks whether the specified DMA stream interrupt has occurred or not.



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



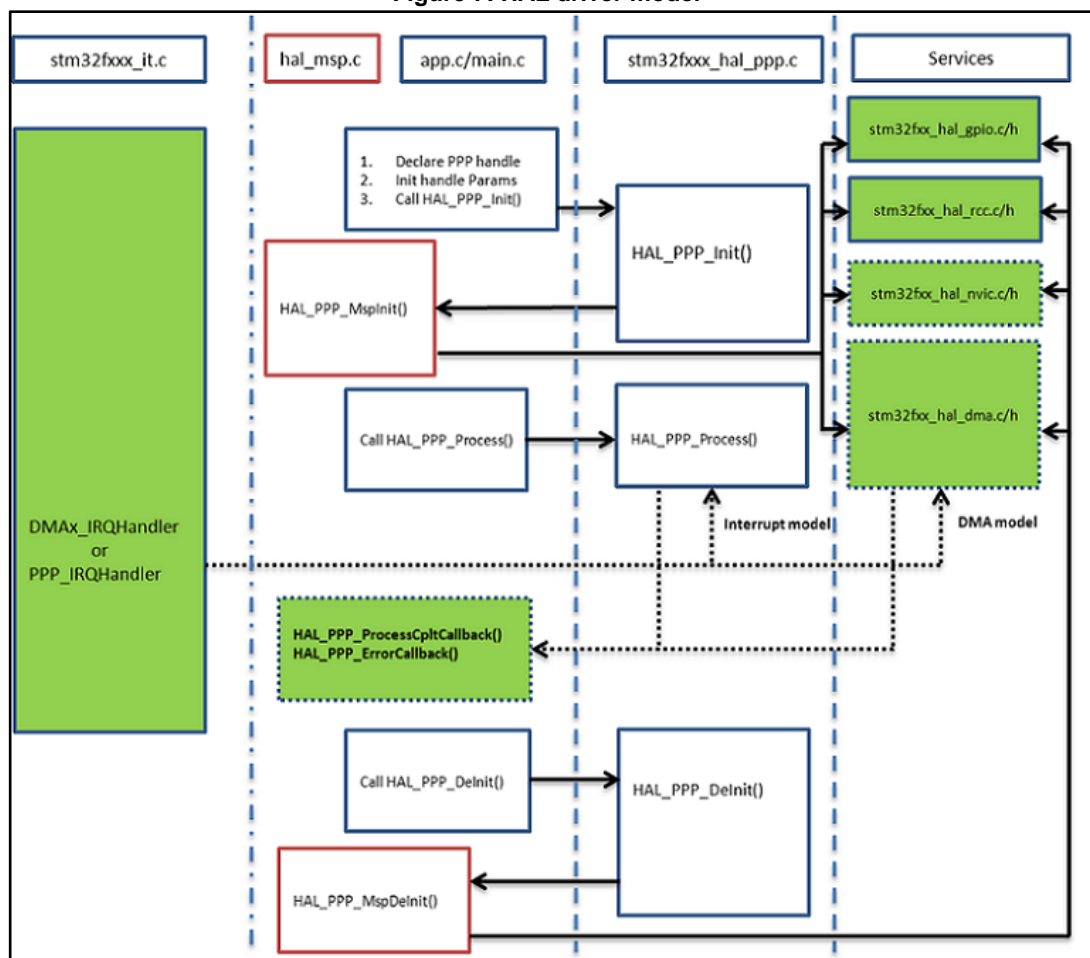
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

## 2.12 How to use HAL drivers

### 2.12.1 HAL usage models

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

Figure 7: HAL driver model



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

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

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

## 2.12.2 HAL initialization

### 2.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file `stm32f7xx_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 is a 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:

```
void SystemClock_Config(void) {
    RCC_ClkInitTypeDef RCC_ClkInitStruct;
    RCC_OscInitTypeDef RCC_OscInitStruct;
    HAL_StatusTypeDef ret = HAL_OK; /* 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 = 432;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 9;
    ret =
    HAL_RCC_OscConfig(&RCC_OscInitStruct);
    if(ret !=
    HAL_OK)
    {
        while(1)
        { ; }
    }
    /* Activate
    the OverDrive to reach the 216 MHz Frequency */
    ret =
    HAL_PWREx_EnableOverDrive();
    if(ret !=
    HAL_OK)
    {
        while(1)
        { ; }
    }
    /* 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;
    ret =
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7);
    if(ret !=
```

```

    HAL_OK)
    {
        while(1)
        { ; } }
    }

```

### 2.12.2.3 HAL MSP initialization process

The peripheral initialization is done through *HAL\_PPP\_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL\_PPP\_MspInit()*.

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

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

```

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

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

```

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

The HAL MSP callback is implemented inside the *stm32f7xx\_hal\_msp.c* file in the user folders. An *stm32f7xx\_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.

*stm32f7xx\_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_t tSize, uint32_t tTimeout)
{
    if((pData == NULL) || (tSize == 0))
    {
        return HAL_ERROR;
    }
    (...) while (data processing is running)
    {
        if( timeout reached )
        {
            return HAL_TIMEOUT;
        }
    }
    (...)
    return HAL_OK; }

```

#### 2.12.3.2 Interrupt mode

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

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

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

To use a process in interrupt mode, *HAL\_PPP\_Process\_IT()* is called in the user file and *HAL\_PPP\_IRQHandler* in *stm32f7xx\_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)
{
}
```

*stm32f7xx\_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 *stm32f7xx\_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL\_PPP\_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
    PPP_TypeDef *Instance; /* Register base address */
    PPP_InitTypeDef Init; /* PPP communication parameters */
    HAL_StateTypeDef State; /* PPP communication state */
    (...)
}
```



```
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
    /* Set User Parameters */
    UartHandle.Init.BaudRate = 9600;
    UartHandle.Init.WordLength = UART_DATABITS_8;
    UartHandle.Init.StopBits = UART_STOPBITS_1;
    UartHandle.Init.Parity = UART_PARITY_NONE;
    UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode = UART_MODE_TX_RX;
    UartHandle.Init.Instance = UART1;
    HAL_UART_Init(&UartHandle);
    (...)
}

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

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

An example of use is illustrated below:

*main.c* file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
    /* Set User 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)
{
}
```

*stm32f7xx\_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 stm32f7xx\_hal\_def.h as 0xFFFFFFFF

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

Example of fixed timeout:

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

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

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

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

### 2.12.4.2 Error management

The HAL drivers implement a check for the following items:

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

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

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

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

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

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

When an error occurs during a peripheral process, *HAL\_PPP\_Process()* returns with a *HAL\_ERROR* status. The HAL PPP driver implements the *HAL\_PPP\_GetError()* to allow retrieving the origin of the error.

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

In all peripheral handles, a *HAL\_PPP\_ErrorTypeDef* is defined and used to store the last error code.

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

```
(...)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

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

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

**HAL\_PPP\_GetError ()** must be used in interrupt mode in the error callback:

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
  ErrorCode = HAL_PPP_GetError (hspi); /* retrieve 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 *stm32f7xx\_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 *stm32f7xx\_hal\_conf.h*:

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

```
#else
#define assert_param(expr) ((void)0)
#endif /* USE_FULL_ASSERT */
```

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

```
#ifndef 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\(\)\*](#)
- [\*HAL\\_EnableFMCMemorySwapping\(\)\*](#)
- [\*HAL\\_DisableFMCMemorySwapping\(\)\*](#)

### 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, and instruction cache through ART accelerator.
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\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_DeInit (void )</b>
Function Description	This function de-Initializes common part of the HAL and stops the systick.
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\_MspDeInit

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>

### 3.1.8 HAL\_InitTick

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. The function is declared as __weak to be overwritten in case of other implementation in user file.</li></ul>

### 3.1.9 HAL\_IncTick

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>

### 3.1.10 HAL\_GetTick

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>

### 3.1.11 HAL\_Delay

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 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 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	<ul style="list-style-type: none"><li>• Device identifier</li></ul>
<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	<ul style="list-style-type: none"><li>• None</li></ul>
<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	<ul style="list-style-type: none"><li>• None</li></ul>
<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	<ul style="list-style-type: none"><li>• None</li></ul>
<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	<ul style="list-style-type: none"><li>• None</li></ul>
<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	<ul style="list-style-type: none"><li>• None</li></ul>
<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	<ul style="list-style-type: none"><li>• None</li></ul>
<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	<ul style="list-style-type: none"><li>• None</li></ul>

- |       |   |
|-------|---|
| Notes | <ul style="list-style-type: none"> <li>The I/O compensation cell can be used only when the device supply voltage ranges from 2.4 to 3.6 V.</li> </ul> |
|-------|---|

### 3.1.24 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> |

### 3.1.25 HAL\_EnableFMCMemorySwapping

- |                      |   |
|----------------------|---|
| Function Name        | <b>void HAL_EnableFMCMemorySwapping (void )</b>   |
| Function Description | Enables the FMC Memory Mapping Swapping.  |
| Return values        | <ul style="list-style-type: none"> <li>None</li> </ul>  |
| Notes                | <ul style="list-style-type: none"> <li>SDRAM is accessible at 0x60000000 and NOR/RAM is accessible at 0xC0000000</li> </ul> |

### 3.1.26 HAL\_DisableFMCMemorySwapping

- |                      |   |
|----------------------|---|
| Function Name        | <b>void HAL_DisableFMCMemorySwapping (void )</b>  |
| Function Description | Disables the FMC Memory Mapping Swapping.   |
| Return values        | <ul style="list-style-type: none"> <li>None</li> </ul>  |
| Notes                | <ul style="list-style-type: none"> <li>SDRAM is accessible at 0xC0000000 (default mapping) and NOR/RAM is accessible at 0x60000000 (default mapping)</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\_LPTIM1  
\_\_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\_I2C4\_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\_LPTIM1  
\_\_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\_I2C4\_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\_FMC  
\_\_HAL\_SYSCFG\_REMAPMEMORY\_FMC\_SDRAM

**HAL Private Constants**

\_\_STM32F7xx\_HAL\_VERSION\_MAIN [31:24] main version  
\_\_STM32F7xx\_HAL\_VERSION\_SUB1 [23:16] sub1 version  
\_\_STM32F7xx\_HAL\_VERSION\_SUB2 [15:8] sub2 version  
\_\_STM32F7xx\_HAL\_VERSION\_RC [7:0] release candidate  
\_\_STM32F7xx\_HAL\_VERSION  
IDCODE\_DEVID\_MASK

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

##### Field Documentation

- ***uint32\_t ADC\_InitTypeDef::ClockPrescaler***  
Select the frequency of the clock to the ADC. The clock is common for all the ADCs. This parameter can be a value of [ADC\\_ClockPrescaler](#)
- ***uint32\_t ADC\_InitTypeDef::Resolution***  
Configures the ADC resolution dual mode. This parameter can be a value of [ADC\\_Resolution](#)
- ***uint32\_t ADC\_InitTypeDef::DataAlign***  
Specifies whether the ADC data alignment is left or right. This parameter can be a value of [ADC\\_data\\_align](#)
- ***uint32\_t ADC\_InitTypeDef::ScanConvMode***  
Specifies whether the conversion is performed in Scan (multi channels) or Single (one channel) mode. This parameter can be set to ENABLE or DISABLE
- ***uint32\_t ADC\_InitTypeDef::EOCSelection***  
Specifies whether the EOC flag is set at the end of single channel conversion or at the end of all conversions. This parameter can be a value of [ADC\\_EOCSelection](#)
- ***uint32\_t ADC\_InitTypeDef::ContinuousConvMode***  
Specifies whether the conversion is performed in Continuous or Single mode. This parameter can be set to ENABLE or DISABLE.
- ***uint32\_t ADC\_InitTypeDef::DMAContinuousRequests***  
Specifies whether the DMA requests is performed in Continuous or in Single mode. This parameter can be set to ENABLE or DISABLE.
- ***uint32\_t ADC\_InitTypeDef::NbrOfConversion***  
Specifies the number of ADC conversions that will be done using the sequencer for regular channel group. This parameter must be a number between Min\_Data = 1 and Max\_Data = 16.

- ***uint32\_t ADC\_InitTypeDef::DiscontinuousConvMode***  
Specifies whether the conversion is performed in Discontinuous or not for regular channels. This parameter can be set to ENABLE or DISABLE.
- ***uint32\_t ADC\_InitTypeDef::NbrOfDiscConversion***  
Specifies the number of ADC discontinuous conversions that will be done using the sequencer for regular channel group. This parameter must be a number between Min\_Data = 1 and Max\_Data = 8.
- ***uint32\_t ADC\_InitTypeDef::ExternalTrigConv***  
Selects the external event used to trigger the conversion start of regular group. If set to ADC\_SOFTWARE\_START, external triggers are disabled. This parameter can be a value of [ADC\\_External\\_trigger\\_Source\\_Regular](#) Note: This parameter can be modified only if there is no conversion is ongoing.
- ***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](#) Note: This parameter can be modified only if there is no conversion is ongoing.

#### 4.1.2 ADC\_HandleTypeDef

##### Data Fields

- ***ADC\_TypeDef \* Instance***
- ***ADC\_InitTypeDef Init***
- ***\_\_IO uint32\_t NbrOfCurrentConversionRank***
- ***DMA\_HandleTypeDef \* DMA\_Handle***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_ADC\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

##### Field Documentation

- ***ADC\_TypeDef\* ADC\_HandleTypeDef::Instance***  
Register base address
- ***ADC\_InitTypeDef ADC\_HandleTypeDef::Init***  
ADC required parameters
- ***\_\_IO uint32\_t ADC\_HandleTypeDef::NbrOfCurrentConversionRank***  
ADC number of current conversion rank
- ***DMA\_HandleTypeDef\* ADC\_HandleTypeDef::DMA\_Handle***  
Pointer DMA Handler
- ***HAL\_LockTypeDef ADC\_HandleTypeDef::Lock***  
ADC locking object
- ***\_\_IO HAL\_ADC\_StateTypeDef ADC\_HandleTypeDef::State***  
ADC communication state
- ***\_\_IO uint32\_t ADC\_HandleTypeDef::ErrorCode***  
ADC Error code

#### 4.1.3 ADC\_ChannelConfTypeDef

**Data Fields**

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

**Field Documentation**

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

#### 4.1.4 ADC\_AnalogWDGConfTypeDef

**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



## 4.2 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:  $V_{REF}(\text{minus}) = V_{IN} = V_{REF}(\text{plus})$ .
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 CRYIP 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_MspDeInit()` (recommended code location) or with other device clock parameters configuration:
      - `HAL_RCC_GetOscConfig(&RCC_OscInitStructure);`
      - `RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI;`
      - `RCC_OscInitStructure.HSIState = RCC_HSI_OFF;` (if not used for system clock)
      - `HAL_RCC_OscConfig(&RCC_OscInitStructure);`
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_DeInit()`.
  - 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\_DeInit\(\)`](#)
- [`HAL\_ADC\_MspInit\(\)`](#)
- [`HAL\_ADC\_MspDeInit\(\)`](#)

### 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	<b>HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)</b>
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_DeInit (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\_MspDeInit

Function Name	<b>void HAL_ADC_MspDeInit (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	<ul style="list-style-type: none"> <li>Caution: This function will stop also injected channels.</li> </ul>

#### 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>

#### 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 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>• 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

- HAL status

#### 4.2.26 HAL\_ADC\_AnalogWDGConfig

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

**Function Description** Configures the analog watchdog.

**Parameters**

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **AnalogWDGConfig**: : pointer to an ADC\_AnalogWDGConfTypeDef structure that contains the configuration information of ADC analog watchdog.

**Return values**

- HAL status

#### 4.2.27 HAL\_ADC\_GetState

**Function Name** `HAL_ADC_StateTypeDef HAL_ADC_GetState (ADC_HandleTypeDef * hadc)`

**Function Description** return the ADC state

**Parameters**

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

**Return values**

- HAL state

#### 4.2.28 HAL\_ADC\_GetError

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

**Function Description** Return the ADC error code.

**Parameters**

- **hadc**: pointer to a ADC\_HandleTypeDef structure that contains the configuration information for the specified ADC.

**Return values**

- ADC Error Code

### 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\_OVR OVR 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 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\_T5\_TRGO

ADC\_EXTERNALTRIGCONV\_T4\_CC4

ADC\_EXTERNALTRIGCONV\_T3\_CC4

ADC\_EXTERNALTRIGCONV\_T8\_TRGO

ADC\_EXTERNALTRIGCONV\_T8\_TRGO2

ADC\_EXTERNALTRIGCONV\_T1\_TRGO

ADC\_EXTERNALTRIGCONV\_T1\_TRGO2

ADC\_EXTERNALTRIGCONV\_T2\_TRGO

ADC\_EXTERNALTRIGCONV\_T4\_TRGO

ADC\_EXTERNALTRIGCONV\_T6\_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**

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.

ADC\_SMPR2

**Parameters:**

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

**Return value:**

- None

**Description:**

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

**Parameters:**

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

**Return value:**

- None

ADC\_SQR3\_RK

**Description:**

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

**Parameters:**

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

**Return value:**

- None

ADC\_SQR2\_RK

**Description:**

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

**Parameters:**

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

**Return value:**

- None

ADC\_SQR1\_RK

**Description:**

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

**Parameters:**

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

**Return value:**

- None

ADC\_CR2\_CONTINUOUS

**Description:**

- Enable ADC continuous conversion mode.

## ADC\_CR1\_DISCONTINUOUS

**Parameters:**

- `_CONTINUOUS_MODE_`: Continuous mode.

**Return value:**

- None

**Description:**

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

**Parameters:**

- `_NBR_DISCONTINUOUSCONV_`: Number of discontinuous conversions.

**Return value:**

- None

## ADC\_CR1\_SCANCONV

**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 AutoInjectedConv*
- *uint32\_t InjectedDiscontinuousConvMode*
- *uint32\_t ExternalTrigInjecConvEdge*
- *uint32\_t ExternalTrigInjecConv*

##### Field Documentation

- *uint32\_t ADC\_InjectionConfTypeDef::InjectedChannel*  
Configure the ADC injected channel. This parameter can be a value of [ADC\\_channels](#)
- *uint32\_t ADC\_InjectionConfTypeDef::InjectedRank*  
The rank in the injected group sequencer. This parameter must be a number between Min\_Data = 1 and Max\_Data = 4.
- *uint32\_t ADC\_InjectionConfTypeDef::InjectedSamplingTime*  
The sample time value to be set for the selected channel. This parameter can be a value of [ADC\\_sampling\\_times](#)
- *uint32\_t ADC\_InjectionConfTypeDef::InjectedOffset*  
Defines the offset to be subtracted from the raw converted data when convert injected channels. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFF.
- *uint32\_t ADC\_InjectionConfTypeDef::InjectedNbrOfConversion*  
Specifies the number of ADC conversions that will be done using the sequencer for injected channel group. This parameter must be a number between Min\_Data = 1 and Max\_Data = 4.
- *uint32\_t ADC\_InjectionConfTypeDef::AutoInjectedConv*  
Enables or disables the selected ADC automatic injected group conversion after regular one.
- *uint32\_t ADC\_InjectionConfTypeDef::InjectedDiscontinuousConvMode*  
Specifies whether the conversion is performed in Discontinuous mode or not for injected channels. This parameter can be set to ENABLE or DISABLE.
- *uint32\_t ADC\_InjectionConfTypeDef::ExternalTrigInjecConvEdge*  
Select the external trigger edge and enable the trigger of an injected channels. This parameter can be a value of [ADCEX\\_External\\_trigger\\_edge\\_Injected](#)
- *uint32\_t ADC\_InjectionConfTypeDef::ExternalTrigInjecConv*  
Select the external event used to trigger the start of conversion of a injected channels. This parameter can be a value of [ADCEX\\_External\\_trigger\\_Source\\_Injected](#)

### 5.1.2 ADC\_MultiModeTypeDef

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

peripheral to memory.

- |               |   |
|---------------|---|
| Return values | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>  |
| Notes         | <ul style="list-style-type: none"> <li>• Caution: This function must be used only with the ADC master.</li> </ul> |

### 5.2.10 HAL\_ADCEx\_MultiModeStop\_DMA

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

### 5.2.11 HAL\_ADCEx\_MultiModeGetValue

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

### 5.2.12 HAL\_ADCEx\_InjectedConvCpltCallback

- |                      |  |
|----------------------|--|
| Function Name        | <b>void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)</b>  |
| Function Description | Injected conversion complete callback in non blocking mode.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hadc</b>: pointer to a 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        | <b>HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)</b>   |
| Function Description | Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hadc</b>: pointer to a 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&amp;1 then 1&amp;3 then 3&amp;2)

ADC\_DMAACCESSMODE\_3 DMA mode 3 enabled (2 / 3 bytes by pairs - 2&amp;1 then 1&amp;3 then 3&amp;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\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T1\_CC4  
ADC\_EXTERNALTRIGINJECCONV\_T2\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T2\_CC1  
ADC\_EXTERNALTRIGINJECCONV\_T3\_CC4  
ADC\_EXTERNALTRIGINJECCONV\_T4\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T8\_CC4  
ADC\_EXTERNALTRIGINJECCONV\_T1\_TRGO2  
ADC\_EXTERNALTRIGINJECCONV\_T8\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T8\_TRGO2  
ADC\_EXTERNALTRIGINJECCONV\_T3\_CC3  
ADC\_EXTERNALTRIGINJECCONV\_T5\_TRGO  
ADC\_EXTERNALTRIGINJECCONV\_T3\_CC1  
ADC\_EXTERNALTRIGINJECCONV\_T6\_TRGO

***ADC Injected Channel Selection***

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\_FilterConfTypeDef::FilterActivation***  
Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- ***uint32\_t CAN\_FilterConfTypeDef::BankNumber***  
Select the start slave bank filter. This parameter must be a number between Min\_Data = 0 and Max\_Data = 28

### 6.1.3 CanTxMsgTypeDef

#### 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:  
`__HAL_RCC_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 `HAL_CAN_Init()` function.
4. Transmit the desired CAN frame using `HAL_CAN_Transmit()` function.
5. Receive a CAN frame using `HAL_CAN_Receive()` function.

#### Polling mode IO operation

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

#### Interrupt mode IO operation

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

#### CAN HAL driver macros list

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

- `__HAL_CAN_ENABLE_IT`: Enable the specified CAN interrupts
- `__HAL_CAN_DISABLE_IT`: Disable the specified CAN interrupts

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



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

## 6.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

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

## 6.2.3 IO operation functions

This section provides functions allowing to:

- Transmit a CAN frame message.
- Receive a CAN frame message.
- Enter CAN peripheral in sleep mode.
- Wake up the CAN peripheral from sleep mode.

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

- None

### 6.2.10 HAL\_CAN\_Transmit

Function Name **HAL\_StatusTypeDef HAL\_CAN\_Transmit (CAN\_HandleTypeDef \* hcan, uint32\_t Timeout)**

Function Description Initiates and transmits a CAN frame message.

Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **Timeout**: Specify Timeout value

Return values

- HAL status

### 6.2.11 HAL\_CAN\_Transmit\_IT

Function Name **HAL\_StatusTypeDef HAL\_CAN\_Transmit\_IT (CAN\_HandleTypeDef \* hcan)**

Function Description Initiates and transmits a CAN frame message.

Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.

Return values

- HAL status

### 6.2.12 HAL\_CAN\_Receive

Function Name **HAL\_StatusTypeDef HAL\_CAN\_Receive (CAN\_HandleTypeDef \* hcan, uint8\_t FIFONumber, uint32\_t Timeout)**

Function Description Receives a correct CAN frame.

Parameters

- **hcan**: pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **FIFONumber**: FIFO Number value
- **Timeout**: Specify Timeout value

Return values

- HAL status

### 6.2.13 HAL\_CAN\_Receive\_IT

Function Name **HAL\_StatusTypeDef HAL\_CAN\_Receive\_IT (CAN\_HandleTypeDef \* hcan, uint8\_t FIFONumber)**

Function Description Receives a correct CAN frame.

Parameters

- **hcan**: Pointer to a CAN\_HandleTypeDef structure that contains the configuration information for the specified CAN.
- **FIFONumber**: Specify the FIFO number

Return values

- HAL status

### 6.2.14 HAL\_CAN\_Sleep

Function Name **HAL\_StatusTypeDef HAL\_CAN\_Sleep (CAN\_HandleTypeDef \* hcan)**

Function Description Enters the Sleep (low power) mode.

Parameters	<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>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
<b>__HAL_CAN_ENABLE_IT</b>	<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>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
<b>__HAL_CAN_DISABLE_IT</b>	<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>

`__HAL_CAN_MSG_PENDING`

- None

**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).

**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

\_\_HAL\_CAN\_CLEAR\_FLAG

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

**Return value:**

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

**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).

**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).

**Description:**

- Release the specified receive FIFO.

**Parameters:**

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

**Return value:**

- None

**Description:**

- Cancel a transmit request.

`__HAL_CAN_GET_IT_SOURCE``__HAL_CAN_TRANSMIT_STATUS``__HAL_CAN_FIFO_RELEASE``__HAL_CAN_CANCEL_TRANSMIT`

`__HAL_CAN_DBG_FREEZE`

**Parameters:**

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

**Return value:**

- None

**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

**CAN Private Constants**

CAN_TIMEOUT_VALUE	
CAN_TXSTATUS_NOMAILBOX	CAN cell did not provide CAN_TxStatus_NoMailBox
CAN_FLAG_MASK	

**CAN Private Macros**

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_STDID
IS_CAN_EXTID
IS_CAN_DLC
IS_CAN_IDTYPE
IS_CAN_RTR
IS_CAN_FIFO

**CAN Receive FIFO Number Constants**

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

**CAN Remote Transmission Request**

CAN_RTR_DATA	Data frame
CAN_RTR_REMOTE	Remote frame

**CAN Synchronisation Jump Width**

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

**CAN Time Quantum in bit segment 1**

---

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

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

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

## 7 HAL CEC Generic Driver

### 7.1 CEC Firmware driver registers structures

#### 7.1.1 CEC\_InitTypeDef

##### Data Fields

- *uint32\_t* **SignalFreeTime**
- *uint32\_t* **Tolerance**
- *uint32\_t* **BRERxStop**
- *uint32\_t* **BREErrorBitGen**
- *uint32\_t* **LBPEErrorBitGen**
- *uint32\_t* **BroadcastMsgNoErrorBitGen**
- *uint32\_t* **SignalFreeTimeOption**
- *uint32\_t* **OwnAddress**
- *uint32\_t* **ListenMode**
- *uint8\_t* **InitiatorAddress**

##### Field Documentation

- ***uint32\_t* CEC\_InitTypeDef::SignalFreeTime**  
Set SFT field, specifies the Signal Free Time. It can be one of [CEC\\_Signal\\_Free\\_Time](#) and belongs to the set {0,...,7} where 0x0 is the default configuration else means 0.5 + (SignalFreeTime - 1) nominal data bit periods
- ***uint32\_t* CEC\_InitTypeDef::Tolerance**  
Set RXTOL bit, specifies the tolerance accepted on the received waveforms, it can be a value of [CEC\\_Tolerance](#) : it is either CEC\_STANDARD\_TOLERANCE or CEC\_EXTENDED\_TOLERANCE
- ***uint32\_t* CEC\_InitTypeDef::BRERxStop**  
Set BRESTP bit [CEC\\_BRERxStop](#) : specifies whether or not a Bit Rising Error stops the reception. CEC\_NO\_RX\_STOP\_ON\_BRE: reception is not stopped.  
CEC\_RX\_STOP\_ON\_BRE: reception is stopped.
- ***uint32\_t* CEC\_InitTypeDef::BREErrorBitGen**  
Set BREGEN bit [CEC\\_BREErrorBitGen](#) : specifies whether or not an Error-Bit is generated on the CEC line upon Bit Rising Error detection.  
CEC\_BRE\_ERRORBIT\_NO\_GENERATION: no error-bit generation.  
CEC\_BRE\_ERRORBIT\_GENERATION: error-bit generation if BRESTP is set.
- ***uint32\_t* CEC\_InitTypeDef::LBPEErrorBitGen**  
Set LBPEGEN bit [CEC\\_LBPEErrorBitGen](#) : specifies whether or not an Error-Bit is generated on the CEC line upon Long Bit Period Error detection.  
CEC\_LBPE\_ERRORBIT\_NO\_GENERATION: no error-bit generation.  
CEC\_LBPE\_ERRORBIT\_GENERATION: error-bit generation.
- ***uint32\_t* CEC\_InitTypeDef::BroadcastMsgNoErrorBitGen**  
Set BRDNOGEN bit [CEC\\_BroadCastMsgErrorBitGen](#) : allows to avoid an Error-Bit generation on the CEC line upon an error detected on a broadcast message. It supersedes BREGEN and LBPEGEN bits for a broadcast message error handling. It can take two values: 1) CEC\_BROADCASTERROR\_ERRORBIT\_GENERATION. a) BRE detection: error-bit generation on the CEC line if BRESTP=CEC\_RX\_STOP\_ON\_BRE and

BREGEN=CEC\_BRE\_ERRORBIT\_NO\_GENERATION. b) LBPE detection: error-bit generation on the CEC line if  
 LBPGEN=CEC\_LBPE\_ERRORBIT\_NO\_GENERATION.2)  
 CEC\_BROADCASTERROR\_NO\_ERRORBIT\_GENERATION. no error-bit generation in case neither a) nor b) are satisfied. Additionally, there is no error-bit generation in case of Short Bit Period Error detection in a broadcast message while LSTN bit is set.

- ***uint32\_t CEC\_InitTypeDef::SignalFreeTimeOption***  
 Set SFTOP bit ***CEC\_SFT\_Option*** : specifies when SFT timer starts.  
 CEC\_SFT\_START\_ON\_TXSOM SFT: timer starts when TXSOM is set by software.  
 CEC\_SFT\_START\_ON\_TX\_RX\_END: SFT timer starts automatically at the end of message transmission/reception.
- ***uint32\_t CEC\_InitTypeDef::OwnAddress***  
 Set OAR field, specifies CEC device address within a 15-bit long field
- ***uint32\_t CEC\_InitTypeDef::ListenMode***  
 Set LSTN bit ***CEC\_Listening\_Mode*** : specifies device listening mode. It can take two values:CEC\_REDUCED\_LISTENING\_MODE: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.CEC\_FULL\_LISTENING\_MODE: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.
- ***uint8\_t CEC\_InitTypeDef::InitiatorAddress***

### 7.1.2 CEC\_HandleTypeDef

#### Data Fields

- ***CEC\_TypeDef \* Instance***
- ***CEC\_InitTypeDef Init***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint32\_t ErrorCode***
- ***HAL\_LockTypeDef Lock***
- ***HAL\_CEC\_StateTypeDef State***

#### Field Documentation

- ***CEC\_TypeDef\* CEC\_HandleTypeDef::Instance***
- ***CEC\_InitTypeDef CEC\_HandleTypeDef::Init***
- ***uint8\_t\* CEC\_HandleTypeDef::pTxBuffPtr***
- ***uint16\_t CEC\_HandleTypeDef::TxXferCount***
- ***uint8\_t\* CEC\_HandleTypeDef::pRxBuffPtr***
- ***uint16\_t CEC\_HandleTypeDef::RxXferSize***
- ***uint32\_t CEC\_HandleTypeDef::ErrorCode***
- ***HAL\_LockTypeDef CEC\_HandleTypeDef::Lock***
- ***HAL\_CEC\_StateTypeDef CEC\_HandleTypeDef::State***

## 7.2 CEC Firmware driver API description

### 7.2.1 How to use this driver

The CEC HAL driver can be used as follow:

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



This API (HAL\_CEC\_Init()) configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_CEC\_MspInit() API.

### 7.2.2 Initialization and Configuration functions

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

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

This section contains the following APIs:

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

### 7.2.3 IO operation functions

This section contains the following APIs:

- [HAL\\_CEC\\_Transmit\(\)](#)
- [HAL\\_CEC\\_Receive\(\)](#)
- [HAL\\_CEC\\_Transmit\\_IT\(\)](#)
- [HAL\\_CEC\\_Receive\\_IT\(\)](#)
- [HAL\\_CEC\\_GetReceivedFrameSize\(\)](#)
- [HAL\\_CEC\\_IRQHandler\(\)](#)
- [HAL\\_CEC\\_TxCpltCallback\(\)](#)
- [HAL\\_CEC\\_RxCpltCallback\(\)](#)
- [HAL\\_CEC\\_ErrorCallback\(\)](#)

## 7.2.4 Peripheral Control function

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

- [HAL\\_CEC\\_GetState\(\)](#) API can be helpful to check in run-time the state of the CEC peripheral.

This section contains the following APIs:

- [HAL\\_CEC\\_GetState\(\)](#)
- [HAL\\_CEC\\_GetError\(\)](#)

## 7.2.5 HAL\_CEC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Init (CEC_HandleTypeDef * hcec)</b>
Function Description	Initializes the CEC mode according to the specified parameters in the CEC_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 7.2.6 HAL\_CEC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_CEC_DeInit (CEC_HandleTypeDef * hcec)</b>
Function Description	DeInitializes the CEC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 7.2.7 HAL\_CEC\_MspltInit

Function Name	<b>void HAL_CEC_MspltInit (CEC_HandleTypeDef * hcec)</b>
Function Description	CEC MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 7.2.8 HAL\_CEC\_MspDeInit

Function Name	<b>void HAL_CEC_MspDeInit (CEC_HandleTypeDef * hcec)</b>
Function Description	CEC MSP DeInit.

Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 7.2.9 HAL\_CEC\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Transmit</b> (CEC_HandleTypeDef * hcec, uint8_t DestinationAddress, uint8_t * pData, uint32_t Size, uint32_t Timeout)
Function Description	Send data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> <li>• <b>DestinationAddress</b>: destination logical address</li> <li>• <b>pData</b>: pointer to input byte data buffer</li> <li>• <b>Size</b>: amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).</li> <li>• <b>Timeout</b>: Timeout duration.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 7.2.10 HAL\_CEC\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Receive</b> (CEC_HandleTypeDef * hcec, uint8_t * pData, uint32_t Timeout)
Function Description	Receive data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> <li>• <b>pData</b>: pointer to received data buffer.</li> <li>• <b>Timeout</b>: Timeout duration. Note that the received data size is not known beforehand, the latter is known when the reception is complete and is stored in hcec-&gt;RxXferSize. hcec-&gt;RxXferSize is the sum of opcodes + operands (0 to 14 operands max). If only a header is received, hcec-&gt;RxXferSize = 0</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 7.2.11 HAL\_CEC\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Transmit_IT</b> (CEC_HandleTypeDef * hcec, uint8_t DestinationAddress, uint8_t * pData, uint32_t Size)
Function Description	Send data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> <li>• <b>DestinationAddress</b>: destination logical address</li> <li>• <b>pData</b>: pointer to input byte data buffer</li> <li>• <b>Size</b>: amount of data to be sent in bytes (without counting the header). 0 means only the header is sent (ping operation). Maximum TX size is 15 bytes (1 opcode and up to 14 operands).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>



**7.2.12 HAL\_CEC\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_CEC_Receive_IT (CEC_HandleTypeDef * hcec, uint8_t * pData)</b>
Function Description	Receive data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> <li>• <b>pData</b>: pointer to received data buffer. Note that the received data size is not known beforehand, the latter is known when the reception is complete and is stored in hcec-&gt;RxXferSize. hcec-&gt;RxXferSize is the sum of opcodes + operands (0 to 14 operands max). If only a header is received, hcec-&gt;RxXferSize = 0</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**7.2.13 HAL\_CEC\_GetReceivedFrameSize**

Function Name	<b>uint32_t HAL_CEC_GetReceivedFrameSize (CEC_HandleTypeDef * hcec)</b>
Function Description	Get size of the received frame.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Frame size</li> </ul>

**7.2.14 HAL\_CEC\_IRQHandler**

Function Name	<b>void HAL_CEC_IRQHandler (CEC_HandleTypeDef * hcec)</b>
Function Description	This function handles CEC interrupt requests.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**7.2.15 HAL\_CEC\_TxCpltCallback**

Function Name	<b>void HAL_CEC_TxCpltCallback (CEC_HandleTypeDef * hcec)</b>
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**7.2.16 HAL\_CEC\_RxCpltCallback**

Function Name	<b>void HAL_CEC_RxCpltCallback (CEC_HandleTypeDef * hcec)</b>
Function Description	Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**7.2.17 HAL\_CEC\_ErrorCallback**

Function Name	<b>void HAL_CEC_ErrorCallback (CEC_HandleTypeDef * hcec)</b>
---------------	--

Function Description	CEC error callbacks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 7.2.18 HAL\_CEC\_GetState

Function Name	<b>HAL_CEC_StateTypeDef HAL_CEC_GetState (CEC_HandleTypeDef * hcec)</b>
Function Description	return the CEC state
Parameters	<ul style="list-style-type: none"> <li>• <b>hcec</b>: CEC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

### 7.2.19 HAL\_CEC\_GetError

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

## 7.3 CEC Firmware driver defines

### 7.3.1 CEC

**CEC all RX or TX errors flags**

CEC\_ISR\_ALL\_ERROR

**CEC Error Bit Generation if Bit Rise Error reported**

CEC\_BRE\_ERRORBIT\_NO\_GENERATION

CEC\_BRE\_ERRORBIT\_GENERATION

**CEC Reception Stop on Error**

CEC\_NO\_RX\_STOP\_ON\_BRE

CEC\_RX\_STOP\_ON\_BRE

**CEC Error Bit Generation on Broadcast message**

CEC\_BROADCASTERROR\_ERRORBIT\_GENERATION

CEC\_BROADCASTERROR\_NO\_ERRORBIT\_GENERATION

**CEC Error Code**

HAL_CEC_ERROR_NONE	no error
HAL_CEC_ERROR_RXOVR	CEC Rx-Overflow
HAL_CEC_ERROR_BRE	CEC Rx Bit Rising Error
HAL_CEC_ERROR_SBPE	CEC Rx Short Bit period Error
HAL_CEC_ERROR_LBPE	CEC Rx Long Bit period Error

HAL_CEC_ERROR_RXACHE	CEC Rx Missing Acknowledge
HAL_CEC_ERROR_ARBLST	CEC Arbitration Lost
HAL_CEC_ERROR_TXUDR	CEC Tx-Buffer Underrun
HAL_CEC_ERROR_TXERR	CEC Tx-Error
HAL_CEC_ERROR_TXACHE	CEC Tx Missing Acknowledge

**CEC Exported Macros**

\_\_HAL\_CEC\_RESET\_HANDLE\_STATE

**Description:**

- Reset CEC handle state.

**Parameters:**

- \_\_HANDLE\_\_: CEC handle.

**Return value:**

- None

\_\_HAL\_CEC\_GET\_FLAG

**Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the CEC Handle.
- \_\_FLAG\_\_: specifies the flag to check.
  - CEC\_FLAG\_TXACHE: Tx Missing acknowledge Error
  - CEC\_FLAG\_TXERR: Tx Error.
  - CEC\_FLAG\_TXUDR: Tx-Buffer Underrun.
  - CEC\_FLAG\_TXEND: End of transmission (successful transmission of the last byte).
  - CEC\_FLAG\_TXBR: Tx-Byte Request.
  - CEC\_FLAG\_ARBLST: Arbitration Lost
  - CEC\_FLAG\_RXACHE: Rx-Missing Acknowledge
  - CEC\_FLAG\_LBPE: Rx Long period Error
  - CEC\_FLAG\_SBPE: Rx Short period Error
  - CEC\_FLAG\_BRE: Rx Bit Rising Error
  - CEC\_FLAG\_RXOVR: Rx Overrun.
  - CEC\_FLAG\_RXEND: End Of Reception.

`__HAL_CEC_CLEAR_FLAG`

- CEC\_FLAG\_RXBR: Rx-Byte Received.

**Return value:**

- ITStatus

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.
- `__FLAG__`: specifies the interrupt/status flag to clear. This parameter can be one of the following values:
  - CEC\_FLAG\_TXACK: Tx Missing acknowledge Error
  - CEC\_FLAG\_TXERR: Tx Error.
  - CEC\_FLAG\_TXUDR: Tx-Buffer Underrun.
  - CEC\_FLAG\_TXEND: End of transmission (successful transmission of the last byte).
  - CEC\_FLAG\_TXBR: Tx-Byte Request.
  - CEC\_FLAG\_ARBLST: Arbitration Lost
  - CEC\_FLAG\_RXACK: Rx-Missing Acknowledge
  - CEC\_FLAG\_LBPE: Rx Long period Error
  - CEC\_FLAG\_SBPE: Rx Short period Error
  - CEC\_FLAG\_BRE: Rx Bit Rising Error
  - CEC\_FLAG\_RXOVR: Rx Overrun.
  - CEC\_FLAG\_RXEND: End Of Reception.
  - CEC\_FLAG\_RXBR: Rx-Byte Received.

**Return value:**

- none

**Description:**

- Enables the specified CEC interrupt.

**Parameters:**`__HAL_CEC_ENABLE_IT`

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

**Return value:**

- none

**Description:**

- Disables the specified CEC interrupt.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.
- `__INTERRUPT__`: specifies the CEC interrupt to disable. This parameter can be one of the following values:
  - `CEC_IT_TXACHE`: Tx Missing acknowledge Error IT Enable
  - `CEC_IT_TXERR`: Tx Error IT Enable

`__HAL_CEC_DISABLE_IT`

- CEC\_IT\_TXUDR: Tx-Buffer Underrun IT Enable
- CEC\_IT\_TXEND: End of transmission IT Enable
- CEC\_IT\_TXBR: Tx-Byte Request IT Enable
- CEC\_IT\_ARBLST: Arbitration Lost IT Enable
- CEC\_IT\_RXACHE: Rx-Missing Acknowledge IT Enable
- CEC\_IT\_LBPE: Rx Long period Error IT Enable
- CEC\_IT\_SBPE: Rx Short period Error IT Enable
- CEC\_IT\_BRE: Rx Bit Rising Error IT Enable
- CEC\_IT\_RXOVR: Rx Overrun IT Enable
- CEC\_IT\_RXEND: End Of Reception IT Enable
- CEC\_IT\_RXBR: Rx-Byte Received IT Enable

**Return value:**

- none

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

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

**Parameters:**

- \_\_HANDLE\_\_: specifies the CEC Handle.
- \_\_INTERRUPT\_\_: specifies the CEC interrupt to check. This parameter can be one of the following values:
  - CEC\_IT\_TXACHE: Tx Missing acknowledge Error IT Enable
  - CEC\_IT\_TXERR: Tx Error IT Enable
  - CEC\_IT\_TXUDR: Tx-Buffer Underrun IT Enable
  - CEC\_IT\_TXEND: End of transmission IT Enable
  - CEC\_IT\_TXBR: Tx-Byte Request IT Enable
  - CEC\_IT\_ARBLST: Arbitration Lost IT Enable
  - CEC\_IT\_RXACHE: Rx-Missing Acknowledge IT

- Enable
- CEC\_IT\_LBPE: Rx Long period Error IT Enable
- CEC\_IT\_SBPE: Rx Short period Error IT Enable
- CEC\_IT\_BRE: Rx Bit Rising Error IT Enable
- CEC\_IT\_RXOVR: Rx Overrun IT Enable
- CEC\_IT\_RXEND: End Of Reception IT Enable
- CEC\_IT\_RXBR: Rx-Byte Received IT Enable

**Return value:**

- FlagStatus

**Description:**

- Enables the CEC device.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none

**Description:**

- Disables the CEC device.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none

**Description:**

- Set Transmission Start flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none

**Description:**

- Set Transmission End flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

`__HAL_CEC_ENABLE``__HAL_CEC_DISABLE``__HAL_CEC_FIRST_BYTE_TX_SET``__HAL_CEC_LAST_BYTE_TX_SET`

`__HAL_CEC_GET_TRANSMISSION_START_FLAG`

**Return value:**

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

**Description:**

- Get Transmission Start flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- FlagStatus

`__HAL_CEC_GET_TRANSMISSION_END_FLAG`

**Description:**

- Get Transmission End flag.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- FlagStatus

`__HAL_CEC_CLEAR_OAR`

**Description:**

- Clear OAR register.

**Parameters:**

- `__HANDLE__`: specifies the CEC Handle.

**Return value:**

- none

`__HAL_CEC_SET_OAR`

**Description:**

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

**Parameters:**

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

**Return value:**

- none

**CEC Flags definition**



CEC\_FLAG\_TXACKE

CEC\_FLAG\_TXERR

CEC\_FLAG\_TXUDR

CEC\_FLAG\_TXEND

CEC\_FLAG\_TXBR

CEC\_FLAG\_ARBLST

CEC\_FLAG\_RXACKE

CEC\_FLAG\_LBPE

CEC\_FLAG\_SBPE

CEC\_FLAG\_BRE

CEC\_FLAG\_RXOVR

CEC\_FLAG\_RXEND

CEC\_FLAG\_RXBR

***CEC all RX errors interrupts enabling flag***

CEC\_IER\_RX\_ALL\_ERR

***CEC all TX errors interrupts enabling flag***

CEC\_IER\_TX\_ALL\_ERR

***CEC Initiator logical address position in message header***

CEC\_INITIATOR\_LSB\_POS

***CEC Interrupts definition***

CEC\_IT\_TXACKE

CEC\_IT\_TXERR

CEC\_IT\_TXUDR

CEC\_IT\_TXEND

CEC\_IT\_TXBR

CEC\_IT\_ARBLST

CEC\_IT\_RXACKE

CEC\_IT\_LBPE

CEC\_IT\_SBPE

CEC\_IT\_BRE

CEC\_IT\_RXOVR

CEC\_IT\_RXEND

CEC\_IT\_RXBR

***CEC Error Bit Generation if Long Bit Period Error reported***

CEC\_LBPE\_ERRORBIT\_NO\_GENERATION

CEC\_LBPE\_ERRORBIT\_GENERATION

***CEC Listening mode option***

CEC\_REDUCED\_LISTENING\_MODE

CEC\_FULL\_LISTENING\_MODE

**CEC Device Own Address position in CEC CFGR register**

CEC\_CFGR\_OAR\_LSB\_POS

**CEC Private Constants**

CEC\_CFGR\_FIELDS

**CEC Private Macros**

IS\_CEC\_SIGNALFREETIME

IS\_CEC\_TOLERANCE

IS\_CEC\_BRERXSTOP

IS\_CEC\_BREERRORBITGEN

IS\_CEC\_LBPEERRORBITGEN

IS\_CEC\_BROADCASTERROR\_NO\_ERRORBIT\_GENERATION

IS\_CEC\_SFTOP

IS\_CEC\_LISTENING\_MODE

IS\_CEC\_OAR\_ADDRESS

**Description:**

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

**Parameters:**

- `__ADDRESS__`: CEC own address.

**Return value:**

- Test: result (TRUE or FALSE).

IS\_CEC\_ADDRESS

**Description:**

- Check CEC initiator or destination logical address setting.

**Parameters:**

- `__ADDRESS__`: CEC initiator or logical address.

**Return value:**

- Test: result (TRUE or FALSE).

---

IS\_CEC\_MSGSIZE

**Description:**

- Check CEC message size.

**Parameters:**

- `__SIZE__`: CEC message size.

**Return value:**

- Test: result (TRUE or FALSE).

***CEC Signal Free Time start option***

CEC\_SFT\_START\_ON\_TXSOM

CEC\_SFT\_START\_ON\_TX\_RX\_END

***CEC Signal Free Time setting parameter***

CEC\_DEFAULT\_SFT

CEC\_0\_5\_BITPERIOD\_SFT

CEC\_1\_5\_BITPERIOD\_SFT

CEC\_2\_5\_BITPERIOD\_SFT

CEC\_3\_5\_BITPERIOD\_SFT

CEC\_4\_5\_BITPERIOD\_SFT

CEC\_5\_5\_BITPERIOD\_SFT

CEC\_6\_5\_BITPERIOD\_SFT

***CEC Receiver Tolerance***

CEC\_STANDARD\_TOLERANCE

CEC\_EXTENDED\_TOLERANCE

## 8 HAL CORTEX Generic Driver

### 8.1 CORTEX Firmware driver registers structures

#### 8.1.1 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_MPU_Access_Cacheable`**
- **`uint8_t MPU_Region_InitTypeDef::IsBufferable`**  
Specifies the bufferable status of the protected region. This parameter can be a value of **`CORTEX_MPU_Access_Bufferable`**

## 8.2 CORTEX Firmware driver API description

### 8.2.1 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-M4 exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using `HAL_NVIC_SetPriorityGrouping()` function according to the following table.
2. Configure the priority of the selected IRQ Channels using `HAL_NVIC_SetPriority()`.
3. Enable the selected IRQ Channels using `HAL_NVIC_EnableIRQ()`.
4. please refer to programming manual for details in how to configure priority. 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 `stm32f7xx_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 0xFFFFF

## 8.2.2 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\(\)\*](#)

## 8.2.3 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\(\)\*](#)

## 8.2.4 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>

## 8.2.5 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 (stm32f7xxx.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>

### 8.2.6 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 (stm32f7xxx.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>

### 8.2.7 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 (stm32f7xxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 8.2.8 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>

### 8.2.9 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>                             |

### 8.2.10 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>   |

### 8.2.11 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> |

### 8.2.12 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 (stm32f7xxxx.h))</li> <li>• <b>PriorityGroup:</b> the priority grouping bits length. This parameter can be one of the following values:<br/>           NVIC_PRIORITYGROUP_0: 0 bits for preemption priority 4 bits for subpriority<br/>           NVIC_PRIORITYGROUP_1: 1 bits for preemption priority 3 bits for subpriority<br/>           NVIC_PRIORITYGROUP_2: 2 bits for preemption priority 2 bits for subpriority<br/>           NVIC_PRIORITYGROUP_3: 3 bits for preemption priority 1 bits for subpriority<br/>           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>   |

### 8.2.13 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 (stm32f7xxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 8.2.14 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 (stm32f7xxx.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>

#### 8.2.15 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 (stm32f7xxx.h))</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 8.2.16 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 (stm32f7xxx.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>

#### 8.2.17 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

### 8.2.18 HAL\_SYSTICK\_IRQHandler

Function Name **void HAL\_SYSTICK\_IRQHandler (void )**

Function Description This function handles SYSTICK interrupt request.

Return values

- None

### 8.2.19 HAL\_SYSTICK\_Callback

Function Name **void HAL\_SYSTICK\_Callback (void )**

Function Description SYSTICK callback.

Return values

- None

## 8.3 CORTEX Firmware driver defines

### 8.3.1 CORTEX

#### *CORTEX Exported Macros*

**\_\_HAL\_CORTEX\_SYSTICKCLK\_CONFIG** **Description:**

- Configures the SysTick clock source.

#### **Parameters:**

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

#### **Return value:**

- None

#### *CORTEX 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\_HARDFULT\_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

## 9 HAL CRC Generic Driver

### 9.1 CRC Firmware driver registers structures

#### 9.1.1 CRC\_InitTypeDef

##### Data Fields

- *uint8\_t DefaultPolynomialUse*
- *uint8\_t DefaultInitValueUse*
- *uint32\_t GeneratingPolynomial*
- *uint32\_t CRCLength*
- *uint32\_t InitValue*
- *uint32\_t InputDataInversionMode*
- *uint32\_t OutputDataInversionMode*

##### Field Documentation

- ***uint8\_t CRC\_InitTypeDef::DefaultPolynomialUse***  
This parameter is a value of [CRC\\_Default\\_Polynomial](#) and indicates if default polynomial is used. If set to DEFAULT\_POLYNOMIAL\_ENABLE, resort to default  $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ . In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT\_POLYNOMIAL\_DISABLE, GeneratingPolynomial and CRCLength fields must be set
- ***uint8\_t CRC\_InitTypeDef::DefaultInitValueUse***  
This parameter is a value of [CRC\\_Default\\_InitValue\\_Use](#) and indicates if default init value is used. If set to DEFAULT\_INIT\_VALUE\_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT\_INIT\_VALUE\_DISABLE, InitValue field must be set
- ***uint32\_t CRC\_InitTypeDef::GeneratingPolynomial***  
Set CRC generating polynomial. 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal representation, e.g., for a polynomial of degree 7,  $X^7 + X^6 + X^5 + X^2 + 1$  is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT\_POLYNOMIAL\_ENABLE
- ***uint32\_t CRC\_InitTypeDef::CRCLength***  
This parameter is a value of [CRC\\_Polynomial\\_Sizes](#) and indicates CRC length. Value can be either one of CRC\_POLYLENGTH\_32B (32-bit CRC) CRC\_POLYLENGTH\_16B (16-bit CRC) CRC\_POLYLENGTH\_8B (8-bit CRC) CRC\_POLYLENGTH\_7B (7-bit CRC)
- ***uint32\_t CRC\_InitTypeDef::InitValue***  
Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT\_INIT\_VALUE\_ENABLE
- ***uint32\_t CRC\_InitTypeDef::InputDataInversionMode***  
This parameter is a value of [CRCEX\\_Input\\_Data\\_Inversion](#) and specifies input data inversion mode. Can be either one of the following values  
CRC\_INPUTDATA\_INVERSION\_NONE no input data inversion  
CRC\_INPUTDATA\_INVERSION\_BYTE byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2  
CRC\_INPUTDATA\_INVERSION\_HALFWORD halfword-wise inversion,

0x1A2B3C4D becomes 0xD458B23C CRC\_INPUTDATA\_INVERSION\_WORD word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- ***uint32\_t CRC\_InitTypeDef::OutputDataInversionMode***  
This parameter is a value of [CRCEx\\_Output\\_Data\\_Inversion](#) and specifies output data (i.e. CRC) inversion mode. Can be either  
CRC\_OUTPUTDATA\_INVERSION\_DISABLE no CRC inversion, or  
CRC\_OUTPUTDATA\_INVERSION\_ENABLE CRC 0x11223344 is converted into 0x22CC4488

## 9.1.2 CRC\_HandleTypeDef

### Data Fields

- ***CRC\_TypeDef \* Instance***
- ***CRC\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_CRC\_StateTypeDef State***
- ***uint32\_t InputDataFormat***

### Field Documentation

- ***CRC\_TypeDef\* CRC\_HandleTypeDef::Instance***  
Register base address
- ***CRC\_InitTypeDef CRC\_HandleTypeDef::Init***  
CRC configuration parameters
- ***HAL\_LockTypeDef CRC\_HandleTypeDef::Lock***  
CRC Locking object
- ***\_\_IO HAL\_CRC\_StateTypeDef CRC\_HandleTypeDef::State***  
CRC communication state
- ***uint32\_t CRC\_HandleTypeDef::InputDataFormat***  
This parameter is a value of [CRC\\_Input\\_Buffer\\_Format](#) and specifies input data format. Can be either CRC\_INPUTDATA\_FORMAT\_BYTES input data is a stream of bytes (8-bit data) CRC\_INPUTDATA\_FORMAT\_HALFWORDS input data is a stream of half-words (16-bit data) CRC\_INPUTDATA\_FORMAT\_WORDS input data is a stream of words (32-bits data) Note that constant CRC\_INPUT\_FORMAT\_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is not one of the three values listed above

## 9.2 CRC Firmware driver API description

### 9.2.1 CRC How to use this driver

1. Enable CRC AHB clock using `__HAL_RCC_CRC_CLK_ENABLE();`
2. Initialize CRC calculator
  - specify generating polynomial (IP default or non-default one)
  - specify initialization value (IP default or non-default one)
  - specify input data format
  - specify input or output data inversion mode if any

3. Use HAL\_CRC\_Accumulate() function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value
4. Use HAL\_CRC\_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

## 9.2.2 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\(\)](#)

## 9.2.3 Peripheral Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [HAL\\_CRC\\_Accumulate\(\)](#)
- [HAL\\_CRC\\_Calculate\(\)](#)

## 9.2.4 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\(\)](#)

## 9.2.5 HAL\_CRC\_Init

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

## 9.2.6 HAL\_CRC\_DeInit

Function Name	HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)
---------------	---



Function Description	DeInitializes the CRC peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 9.2.7 HAL\_CRC\_Mspltinit

Function Name	<b>void HAL_CRC_Mspltinit (CRC_HandleTypeDef * hcrc)</b>
Function Description	Initializes the CRC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 9.2.8 HAL\_CRC\_MspDeInit

Function Name	<b>void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)</b>
Function Description	DeInitializes the CRC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 9.2.9 HAL\_CRC\_Accumulate

Function Name	<b>uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)</b>
Function Description	Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> <li>• <b>pBuffer:</b> pointer to the input data buffer, exact input data format is provided by hcrc-&gt;InputDataFormat.</li> <li>• <b>BufferLength:</b> input data buffer length</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• uint32_t CRC (returned value LSBs for CRC shorter than 32 bits)</li> </ul>

### 9.2.10 HAL\_CRC\_Calculate

Function Name	<b>uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)</b>
Function Description	Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> <li>• <b>pBuffer:</b> pointer to the input data buffer, exact input data format is provided by hcrc-&gt;InputDataFormat.</li> <li>• <b>BufferLength:</b> input data buffer length</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• uint32_t CRC (returned value LSBs for CRC shorter than 32 bits)</li> </ul>

### 9.2.11 HAL\_CRC\_GetState



Function Name	<b>HAL_CRC_StateTypeDef HAL_CRC_GetState</b> (CRC_HandleTypeDef * hcrc)
Function Description	Returns the CRC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc</b>: CRC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

## 9.3 CRC Firmware driver defines

### 9.3.1 CRC

**Default CRC computation initialization value**

DEFAULT\_CRC\_INITVALUE

**Indicates whether or not default init value is used**

DEFAULT\_INIT\_VALUE\_ENABLE

DEFAULT\_INIT\_VALUE\_DISABLE

**Indicates whether or not default polynomial is used**

DEFAULT\_POLYNOMIAL\_ENABLE

DEFAULT\_POLYNOMIAL\_DISABLE

**Default CRC generating polynomial**

DEFAULT\_CRC32\_POLY

**CRC Exported Functions**

HAL\_CRC\_Input\_Data\_Reverse

HAL\_CRC\_Output\_Data\_Reverse

**CRC exported macros**

\_\_HAL\_CRC\_RESET\_HANDLE\_STATE

**Description:**

- Reset CRC handle state.

**Parameters:**

- **\_\_HANDLE\_\_**: CRC handle.

**Return value:**

- None

\_\_HAL\_CRC\_DR\_RESET

**Description:**

- Reset CRC Data Register.

**Parameters:**

- **\_\_HANDLE\_\_**: CRC handle

**Return value:**

- None.

\_\_HAL\_CRC\_INITIALCRCVALUE\_CONFIG

**Description:**

- Set CRC INIT non-default value.

\_\_HAL\_CRC\_SET\_IDR

**Parameters:**

- \_\_HANDLE\_\_ : CRC handle
- \_\_INIT\_\_ : 32-bit initial value

**Return value:**

- None.

**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

**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

\_\_HAL\_CRC\_GET\_IDR

**CRC input buffer format**

CRC\_INPUTDATA\_FORMAT\_UNDEFINED

CRC\_INPUTDATA\_FORMAT\_BYTES

CRC\_INPUTDATA\_FORMAT\_HALFWORDS

CRC\_INPUTDATA\_FORMAT\_WORDS

**Polynomial sizes to configure the IP**

CRC\_POLYLENGTH\_32B

CRC\_POLYLENGTH\_16B

CRC\_POLYLENGTH\_8B

CRC\_POLYLENGTH\_7B

**CRC polynomial possible sizes actual definitions**

HAL\_CRC\_LENGTH\_32B

HAL\_CRC\_LENGTH\_16B

HAL\_CRC\_LENGTH\_8B

HAL\_CRC\_LENGTH\_7B

**CRC Private Macros**

IS\_DEFAULT\_POLYNOMIAL

IS\_DEFAULT\_INIT\_VALUE

IS\_CRC\_POL\_LENGTH

IS\_CRC\_INPUTDATA\_FORMAT

## 10 HAL CRC Extension Driver

### 10.1 CRCEX Firmware driver API description

#### 10.1.1 CRC Extended features functions

This subsection provides function allowing to:

- Set CRC polynomial if different from default one.

This section contains the following APIs:

- [HAL\\_CRCEX\\_Polynomial\\_Set\(\)](#)
- [HAL\\_CRCEX\\_Input\\_Data\\_Reverse\(\)](#)
- [HAL\\_CRCEX\\_Output\\_Data\\_Reverse\(\)](#)

#### 10.1.2 HAL\_CRCEX\_Polynomial\_Set

Function Name	<b>HAL_StatusTypeDef HAL_CRCEX_Polynomial_Set</b> (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)
Function Description	Initializes the CRC polynomial if different from default one.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> <li>• <b>Pol:</b> CRC generating polynomial (7, 8, 16 or 32-bit long) This parameter is written in normal representation, e.g. for a polynomial of degree 7, <math>X^7 + X^6 + X^5 + X^2 + 1</math> is written 0x65 for a polynomial of degree 16, <math>X^{16} + X^{12} + X^5 + 1</math> is written 0x1021</li> <li>• <b>PolyLength:</b> CRC polynomial length This parameter can be one of the following values: CRC_POLYLENGTH_7B: 7-bit long CRC (generating polynomial of degree 7) CRC_POLYLENGTH_8B: 8-bit long CRC (generating polynomial of degree 8) CRC_POLYLENGTH_16B: 16-bit long CRC (generating polynomial of degree 16) CRC_POLYLENGTH_32B: 32-bit long CRC (generating polynomial of degree 32)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 10.1.3 HAL\_CRCEX\_Input\_Data\_Reverse

Function Name	<b>HAL_StatusTypeDef HAL_CRCEX_Input_Data_Reverse</b> (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)
Function Description	Set the Reverse Input data mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrc:</b> CRC handle</li> <li>• <b>InputReverseMode:</b> Input Data inversion mode This parameter can be one of the following values: CRC_INPUTDATA_INVERSION_NONE: no change in bit order (default value) CRC_INPUTDATA_INVERSION_BYTE: Byte-wise bit reversal CRC_INPUTDATA_INVERSION_HALFWORD: HalfWord-wise bit</li> </ul>

reversalCRC\_INPUTDATA\_INVERSION\_WORD: Word-wise  
bit reversal

## Return values

- HAL status

### 10.1.4 HAL\_CRCEx\_Output\_Data\_Reverse

## Function Name

**HAL\_StatusTypeDef HAL\_CRCEx\_Output\_Data\_Reverse**  
(CRC\_HandleTypeDef \* hcrc, uint32\_t OutputReverseMode)

## Function Description

Set the Reverse Output data mode.

## Parameters

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

## Return values

- HAL status

## 10.2 CRCEx Firmware driver defines

### 10.2.1 CRCEx

**CRC Extended exported macros**

**\_\_HAL\_CRC\_OUTPUTREVERSAL\_ENABLE**

## Description:

- Set CRC output reversal.

## Parameters:

- **\_\_HANDLE\_\_**: : CRC handle

## Return value:

- None.

**\_\_HAL\_CRC\_OUTPUTREVERSAL\_DISABLE**

## Description:

- Unset CRC output reversal.

## Parameters:

- **\_\_HANDLE\_\_**: : CRC handle

## Return value:

- None.

**\_\_HAL\_CRC\_POLYNOMIAL\_CONFIG**

## Description:

- Set CRC non-default polynomial.

## Parameters:

- **\_\_HANDLE\_\_**: : CRC handle
- **\_\_POLYNOMIAL\_\_**: 7, 8, 16 or 32-bit polynomial

## Return value:

- None.

***CRC Extended input data inversion modes***

CRC\_INPUTDATA\_INVERSION\_NONE

CRC\_INPUTDATA\_INVERSION\_BYTE

CRC\_INPUTDATA\_INVERSION\_HALFWORD

CRC\_INPUTDATA\_INVERSION\_WORD

IS\_CRC\_INPUTDATA\_INVERSION\_MODE

***CRC Extended output data inversion modes***

CRC\_OUTPUTDATA\_INVERSION\_DISABLE

CRC\_OUTPUTDATA\_INVERSION\_ENABLE

IS\_CRC\_OUTPUTDATA\_INVERSION\_MODE

## 11 HAL CRYPT Generic Driver

### 11.1 CRYPT Firmware driver registers structures

#### 11.1.1 CRYPT\_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* **CRYPT\_InitTypeDef::DataType**  
32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of [CRYPT\\_Data\\_Type](#)
- *uint32\_t* **CRYPT\_InitTypeDef::KeySize**  
Used only in AES mode only : 128, 192 or 256 bit key length. This parameter can be a value of [CRYPT\\_Key\\_Size](#)
- *uint8\_t*\* **CRYPT\_InitTypeDef::pKey**  
The key used for encryption/decryption
- *uint8\_t*\* **CRYPT\_InitTypeDef::pInitVect**  
The initialization vector used also as initialization counter in CTR mode
- *uint8\_t* **CRYPT\_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* **CRYPT\_InitTypeDef::TagSize**  
The size of returned authentication TAG. This parameter is used only in AES-128/192/256 encryption/decryption CCM mode
- *uint8\_t*\* **CRYPT\_InitTypeDef::Header**  
The header used in GCM and CCM modes
- *uint32\_t* **CRYPT\_InitTypeDef::HeaderSize**  
The size of header buffer in bytes
- *uint8\_t*\* **CRYPT\_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

#### 11.1.2 CRYPT\_HandleTypeDef

##### Data Fields



- ***CRYPT\_TypeDef \* Instance***
- ***CRYPT\_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\_CRYPT\_STATETTypeDef State***

#### Field Documentation

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

## 11.2 CRYPT Firmware driver API description

### 11.2.1 How to use this driver

The CRYPT HAL driver can be used as follows:

1. Initialize the CRYPT low level resources by implementing the `HAL_CRYPT_MspInit()`:
  - a. Enable the CRYPT interface clock using `__HAL_RCC_CRYPT_CLK_ENABLE()`
  - b. In case of using interrupts (e.g. `HAL_CRYPT_AESECBCB_Encrypt_IT()`)
    - Configure the CRYPT interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the CRYPT IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In CRYPT IRQ handler, call `HAL_CRYPT_IRQHandler()`

- c. In case of using DMA to control data transfer (e.g. `HAL_CRYPT_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 CRYPT 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 CRYPT HAL using `HAL_CRYPT_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_CRYPT_AESCBC_Encrypt()`
  - b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. `HAL_CRYPT_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_CRYPT_AESCBC_Encrypt_DMA()`
4. When the processing function is called at first time after `HAL_CRYPT_Init()` the CRYPT 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_CRYPT_Init()` then the processing function.
5. Call `HAL_CRYPT_DeInit()` to deinitialize the CRYPT peripheral.

### 11.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRYPT according to the specified parameters in the `CRYPT_InitTypeDef` and creates the associated handle
- DeInitialize the CRYPT peripheral
- Initialize the CRYPT MSP
- DeInitialize CRYPT MSP

This section contains the following APIs:

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

### 11.2.3 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES-128/192/256 using chaining modes
- Decrypt cyphertext 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\\_CRYPT\\_AESECB\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCBC\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCTR\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESECB\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCBC\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCTR\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESECB\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCBC\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCTR\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESECB\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCBC\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCTR\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESECB\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCBC\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCTR\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESECB\\_Decrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCBC\\_Decrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_AESCTR\\_Decrypt\\_DMA\(\)\*](#)

#### 11.2.4 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\\_CRYPT\\_DESECB\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESECB\\_Decrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_DESCBC\\_Decrypt\\_DMA\(\)\*](#)

#### 11.2.5 TDES processing functions

This section provides functions allowing to:

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

- Decrypt cyphertext 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\\_CRYPT\\_TDESECB\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Encrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Decrypt\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Encrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Decrypt\\_IT\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Encrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESECB\\_Decrypt\\_DMA\(\)\*](#)
- [\*HAL\\_CRYPT\\_TDESCBC\\_Decrypt\\_DMA\(\)\*](#)

### 11.2.6 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\\_CRYPT\\_InCpltCallback\(\)\*](#)
- [\*HAL\\_CRYPT\\_OutCpltCallback\(\)\*](#)
- [\*HAL\\_CRYPT\\_ErrorCallback\(\)\*](#)

### 11.2.7 CRYPT IRQ handler management

This section provides CRYPT IRQ handler function.

This section contains the following APIs:

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

### 11.2.8 Peripheral State functions

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

This section contains the following APIs:

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

### 11.2.9 HAL\_CRYPT\_Init

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_Init (CRYPT_HandleTypeDef * hcrypt)</b>
Function Description	Initializes the CRYPT according to the specified parameters in the CRYPT_InitTypeDef and creates the associated handle.

Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.10 HAL\_CRYPT\_DeInit

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

### 11.2.11 HAL\_CRYPT\_MspsInit

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

### 11.2.12 HAL\_CRYPT\_MspDeInit

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

### 11.2.13 HAL\_CRYPT\_AESECBC\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESECBC_Encrypt (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</b>
Function Description	Initializes the CRYPT peripheral in AES ECB encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.14 HAL\_CRYPT\_AESCBC\_Encrypt

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

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

Function Description	Initializes the CRYPT peripheral in AES CBC encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.15 HAL\_CRYPT\_AESCTR\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCTR_Encrypt</b> (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES CTR encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.16 HAL\_CRYPT\_AESECBC\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESECBC_Decrypt</b> (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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>

### 11.2.17 HAL\_CRYPT\_AESCBC\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCBC_Decrypt</b> (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES ECB decryption mode then

decrypted pCypherData.

Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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>

### 11.2.18 HAL\_CRYPT\_AESCTR\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCTR_Decrypt</b> (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES CTR decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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>

### 11.2.19 HAL\_CRYPT\_AESECBC\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESECBC_Encrypt_IT</b> (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES ECB encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.20 HAL\_CRYPT\_AESCBC\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCBC_Encrypt_IT</b> (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES CBC encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer
- Return values
- HAL status

### 11.2.21 HAL\_CRYPT\_AESCTR\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCTR_Encrypt_IT</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES CTR encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.22 HAL\_CRYPT\_AESECB\_Decrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESECB_Decrypt_IT</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES ECB decryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData:</b> Pointer to the cyphertext 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>

### 11.2.23 HAL\_CRYPT\_AESCBC\_Decrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCBC_Decrypt_IT</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES CBC decryption mode using IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData:</b> Pointer to the cyphertext 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>



**11.2.24 HAL\_CRYPT\_AESCTR\_Decrypt\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCTR_Decrypt_IT</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES CTR decryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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>

**11.2.25 HAL\_CRYPT\_AESECB\_Encrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESECB_Encrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES ECB encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**11.2.26 HAL\_CRYPT\_AESCBC\_Encrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCBC_Encrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES CBC encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**11.2.27 HAL\_CRYPT\_AESCTR\_Encrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_AESCTR_Encrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES CTR encryption mode using

DMA.

- |               |   |
|---------------|---|
| Parameters    | <ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul> |
| Return values | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>  |

### 11.2.28 HAL\_CRYPT\_AESECBC\_Decrypt\_DMA

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYPT_AESECBC_Decrypt_DMA</b><br>(CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)  |
| Function Description | Initializes the CRYPT peripheral in AES ECB decryption mode using DMA.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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>   |

### 11.2.29 HAL\_CRYPT\_AESCBC\_Decrypt\_DMA

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYPT_AESCBC_Decrypt_DMA</b><br>(CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)   |
| Function Description | Initializes the CRYPT peripheral in AES CBC encryption mode using DMA.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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>   |

### 11.2.30 HAL\_CRYPT\_AESCTR\_Decrypt\_DMA

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYPT_AESCTR_Decrypt_DMA</b><br>(CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)   |
| Function Description | Initializes the CRYPT peripheral in AES CTR decryption mode using DMA.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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
- HAL status

### 11.2.31 HAL\_CRYPT\_DESECB\_Encrypt

**Function Name** HAL\_StatusTypeDef HAL\_CRYPT\_DESECB\_Encrypt (CRYPT\_HandleTypeDef \* hcryp, uint8\_t \* pPlainData, uint16\_t Size, uint8\_t \* pCypherData, uint32\_t Timeout)

**Function Description** Initializes the CRYPT peripheral in DES ECB encryption mode.

- Parameters**
- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module
  - **pPlainData:** Pointer to the plaintext buffer
  - **Size:** Length of the plaintext buffer, must be a multiple of 8
  - **pCypherData:** Pointer to the cyphertext buffer
  - **Timeout:** Specify Timeout value

- Return values**
- HAL status

### 11.2.32 HAL\_CRYPT\_DESECB\_Decrypt

**Function Name** HAL\_StatusTypeDef HAL\_CRYPT\_DESECB\_Decrypt (CRYPT\_HandleTypeDef \* hcryp, uint8\_t \* pPlainData, uint16\_t Size, uint8\_t \* pCypherData, uint32\_t Timeout)

**Function Description** Initializes the CRYPT peripheral in DES ECB decryption mode.

- Parameters**
- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module
  - **pPlainData:** Pointer to the plaintext buffer
  - **Size:** Length of the plaintext buffer, must be a multiple of 8
  - **pCypherData:** Pointer to the cyphertext buffer
  - **Timeout:** Specify Timeout value

- Return values**
- HAL status

### 11.2.33 HAL\_CRYPT\_DESCBC\_Encrypt

**Function Name** HAL\_StatusTypeDef HAL\_CRYPT\_DESCBC\_Encrypt (CRYPT\_HandleTypeDef \* hcryp, uint8\_t \* pPlainData, uint16\_t Size, uint8\_t \* pCypherData, uint32\_t Timeout)

**Function Description** Initializes the CRYPT peripheral in DES CBC encryption mode.

- Parameters**
- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module
  - **pPlainData:** Pointer to the plaintext buffer
  - **Size:** Length of the plaintext buffer, must be a multiple of 8
  - **pCypherData:** Pointer to the cyphertext buffer
  - **Timeout:** Specify Timeout value

- Return values**
- HAL status

### 11.2.34 HAL\_CRYPT\_DESCBC\_Decrypt

**Function Name** HAL\_StatusTypeDef HAL\_CRYPT\_DESCBC\_Decrypt (CRYPT\_HandleTypeDef \* hcryp, uint8\_t \* pPlainData, uint16\_t

	<b>Size, uint8_t * pCypherData, uint32_t Timeout)</b>
Function Description	Initializes the CRYPT peripheral in DES ECB decryption mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext 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>11.2.35 HAL_CRYPT_DESECB_Encrypt_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_DESECB_Encrypt_IT (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYPT peripheral in DES ECB encryption mode using IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>11.2.36 HAL_CRYPT_DESCBC_Encrypt_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_DESCBC_Encrypt_IT (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYPT peripheral in DES CBC encryption mode using interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>11.2.37 HAL_CRYPT_DESECB_Decrypt_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_DESECB_Decrypt_IT (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYPT peripheral in DES ECB decryption mode using IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pPlainData</b>: Pointer to the plaintext buffer</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 8</li> <li>• <b>pCypherData:</b> Pointer to the cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>11.2.38 HAL_CRYPT_DESCBC_Decrypt_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_DESCBC_Decrypt_IT</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in DES ECB decryption mode using interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>11.2.39 HAL_CRYPT_DESECB_Encrypt_DMA</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_DESECB_Encrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in DES ECB encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>11.2.40 HAL_CRYPT_DESCBC_Encrypt_DMA</b>	
Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_DESCBC_Encrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in DES CBC encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>11.2.41 HAL_CRYPT_DESECB_Decrypt_DMA</b>	

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_DESECB_Decrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in DES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 11.2.42 HAL\_CRYPT\_DESCBC\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_DESCBC_Decrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in DES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 11.2.43 HAL\_CRYPT\_TDESECB\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_TDESECB_Encrypt</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in TDES ECB encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 11.2.44 HAL\_CRYPT\_TDESECB\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_TDESECB_Decrypt</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in TDES ECB decryption mode then decrypted pCypherData.

Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.45 HAL\_CRYPT\_TDESCBC\_Encrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_TDESCBC_Encrypt</b> (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in TDES CBC encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.46 HAL\_CRYPT\_TDESCBC\_Decrypt

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

### 11.2.47 HAL\_CRYPT\_TDESECB\_Encrypt\_IT

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



- **pCypherData:** Pointer to the cyphertext buffer
- Return values
  - HAL status

### 11.2.48 HAL\_CRYPT\_TDESCBC\_Encrypt\_IT

- Function Name** HAL\_StatusTypeDef HAL\_CRYPT\_TDESCBC\_Encrypt\_IT (CRYPT\_HandleTypeDef \* hcrypt, uint8\_t \* pPlainData, uint16\_t Size, uint8\_t \* pCypherData)
- Function Description** Initializes the CRYPT peripheral in TDES CBC encryption mode.
- Parameters**
- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module
  - **pPlainData:** Pointer to the plaintext buffer
  - **Size:** Length of the plaintext buffer, must be a multiple of 8
  - **pCypherData:** Pointer to the cyphertext buffer
- Return values**
- HAL status

### 11.2.49 HAL\_CRYPT\_TDESECB\_Decrypt\_IT

- Function Name** HAL\_StatusTypeDef HAL\_CRYPT\_TDESECB\_Decrypt\_IT (CRYPT\_HandleTypeDef \* hcrypt, uint8\_t \* pCypherData, uint16\_t Size, uint8\_t \* pPlainData)
- Function Description** Initializes the CRYPT peripheral in TDES ECB decryption mode.
- Parameters**
- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module
  - **pPlainData:** Pointer to the plaintext buffer
  - **Size:** Length of the plaintext buffer, must be a multiple of 8
  - **pCypherData:** Pointer to the cyphertext buffer
- Return values**
- HAL status

### 11.2.50 HAL\_CRYPT\_TDESCBC\_Decrypt\_IT

- Function Name** HAL\_StatusTypeDef HAL\_CRYPT\_TDESCBC\_Decrypt\_IT (CRYPT\_HandleTypeDef \* hcrypt, uint8\_t \* pCypherData, uint16\_t Size, uint8\_t \* pPlainData)
- Function Description** Initializes the CRYPT peripheral in TDES CBC decryption mode.
- Parameters**
- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module
  - **pCypherData:** Pointer to the cyphertext buffer
  - **Size:** Length of the plaintext buffer, must be a multiple of 8
  - **pPlainData:** Pointer to the plaintext buffer
- Return values**
- HAL status

### 11.2.51 HAL\_CRYPT\_TDESECB\_Encrypt\_DMA

- Function Name** HAL\_StatusTypeDef HAL\_CRYPT\_TDESECB\_Encrypt\_DMA (CRYPT\_HandleTypeDef \* hcrypt, uint8\_t \* pPlainData, uint16\_t Size, uint8\_t \* pCypherData)
- Function Description** Initializes the CRYPT peripheral in TDES ECB encryption mode



using DMA.

Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.52 HAL\_CRYPT\_TDESCBC\_Encrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_TDESCBC_Encrypt_DMA (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYPT peripheral in TDES CBC encryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.53 HAL\_CRYPT\_TDESECB\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_TDESECB_Decrypt_DMA (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYPT peripheral in TDES ECB decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 11.2.54 HAL\_CRYPT\_TDESCBC\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYPT_TDESCBC_Decrypt_DMA (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)</b>
Function Description	Initializes the CRYPT peripheral in TDES CBC decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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

- HAL status

### 11.2.55 HAL\_CRYPT\_InCpltCallback

Function Name **void HAL\_CRYPT\_InCpltCallback (CRYPT\_HandleTypeDef \* hcrypt)**

Function Description Input FIFO transfer completed callbacks.

Parameters

- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module

Return values

- None

### 11.2.56 HAL\_CRYPT\_OutCpltCallback

Function Name **void HAL\_CRYPT\_OutCpltCallback (CRYPT\_HandleTypeDef \* hcrypt)**

Function Description Output FIFO transfer completed callbacks.

Parameters

- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module

Return values

- None

### 11.2.57 HAL\_CRYPT\_ErrorCallback

Function Name **void HAL\_CRYPT\_ErrorCallback (CRYPT\_HandleTypeDef \* hcrypt)**

Function Description CRYPT error callbacks.

Parameters

- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module

Return values

- None

### 11.2.58 HAL\_CRYPT\_IRQHandler

Function Name **void HAL\_CRYPT\_IRQHandler (CRYPT\_HandleTypeDef \* hcrypt)**

Function Description This function handles CRYPT interrupt request.

Parameters

- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module

Return values

- None

### 11.2.59 HAL\_CRYPT\_GetState

Function Name **HAL\_CRYPT\_STATTypeDef HAL\_CRYPT\_GetState (CRYPT\_HandleTypeDef \* hcrypt)**

Function Description Returns the CRYPT state.

Parameters

- **hcryp:** pointer to a CRYPT\_HandleTypeDef structure that contains the configuration information for CRYPT module

Return values

- HAL state

## 11.3 CRYP Firmware driver defines

### 11.3.1 CRYP

#### **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**

<code>__HAL_CRYPT_RESET_HANDLE_STATE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Reset CRYPT handle state.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: specifies the CRYPT handle.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_CRYPT_ENABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Enable/Disable the CRYPT peripheral.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: specifies the CRYPT handle.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_CRYPT_DISABLE</code>	
<code>__HAL_CRYPT_FIFO_FLUSH</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Flush the data FIFO.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: specifies the CRYPT handle.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_CRYPT_SET_MODE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Set the algorithm mode: AES-ECB, AES-CBC, AES-CTR, DES-ECB, DES-CBC.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: specifies the CRYPT handle.</li><li><code>MODE</code>: The algorithm mode.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>None</li></ul>
<code>__HAL_CRYPT_GET_FLAG</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>Check whether the specified CRYPT flag is set or not.</li></ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"><li><code>__HANDLE__</code>: specifies the CRYPT handle.</li><li><code>__FLAG__</code>: specifies the flag to check. This parameter can be one of the following values:</li></ul>

- CRYPT\_FLAG\_BUSY: The CRYPT core is currently processing a block of data or a key preparation (for AES decryption).
- CRYPT\_FLAG\_IFEM: Input FIFO is empty
- CRYPT\_FLAG\_IFNF: Input FIFO is not full
- CRYPT\_FLAG\_INRIS: Input FIFO service raw interrupt is pending
- CRYPT\_FLAG\_OFNE: Output FIFO is not empty
- CRYPT\_FLAG\_OFFU: Output FIFO is full
- CRYPT\_FLAG\_OUTRIS: Output FIFO service raw interrupt is pending

**Return value:**

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

**Description:**

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

**Parameters:**

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

**Return value:**

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

**Description:**

- Enable the CRYPT interrupt.

**Parameters:**

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

**Return value:**

- None

**Description:**

- Disable the CRYPT interrupt.

\_\_HAL\_CRYPT\_GET\_IT

\_\_HAL\_CRYPT\_ENABLE\_IT

\_\_HAL\_CRYPT\_DISABLE\_IT

**Parameters:**

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

**Return value:**

- None

**CRYPT Key Size**`CRYPT_KEYSIZE_128B``CRYPT_KEYSIZE_192B``CRYPT_KEYSIZE_256B`**CRYPT Private Constants**`CRYPT_FLAG_MASK`**CRYPT Private define**`CRYPT_TIMEOUT_VALUE`**CRYPT Private Macros**`IS_CRYPT_KEYSIZE``IS_CRYPT_DATATYPE`

## 12 HAL CRYPT Extension Driver

### 12.1 CRYPEX Firmware driver API description

#### 12.1.1 How to use this driver

The CRYPT Extension HAL driver can be used as follows:

1. Initialize the CRYPT low level resources by implementing the `HAL_CRYPT_MspInit()`:
  - a. Enable the CRYPT interface clock using `__HAL_RCC_CRYPT_CLK_ENABLE()`
  - b. In case of using interrupts (e.g. `HAL_CRYPEX_AESGCM_Encrypt_IT()`)
    - Configure the CRYPT interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the CRYPT IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In CRYPT IRQ handler, call `HAL_CRYPT_IRQHandler()`
  - c. In case of using DMA to control data transfer (e.g. `HAL_AES_ECB_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 CRYPT 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 CRYPT HAL using `HAL_CRYPT_Init()`. This function configures mainly:
  - a. The data type: 1-bit, 8-bit, 16-bit and 32-bit
  - b. The key size: 128, 192 and 256. This parameter is relevant only for AES
  - c. The encryption/decryption key. Its size depends on the algorithm used for encryption/decryption
  - d. The initialization vector (counter). It is not used ECB mode.
3. Three processing (encryption/decryption) functions are available:
  - a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished e.g. `HAL_CRYPEX_AESGCM_Encrypt()`
  - b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt e.g. `HAL_CRYPEX_AESGCM_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_CRYPEX_AESGCM_Encrypt_DMA()`
4. When the processing function is called at first time after `HAL_CRYPT_Init()` the CRYPT 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_CRYPT_Init()` then the processing function.
5. In AES-GCM and AES-CCM modes are an authenticated encryption algorithms which provide authentication messages. `HAL_AES_GCM_Finish()` and `HAL_AES_CCM_Finish()` are used to provide those authentication messages. Call those functions after the processing ones (polling, interrupt or DMA). e.g. in AES-CCM mode call `HAL_CRYPEX_AESCCM_Encrypt()` to encrypt the plain data then call `HAL_CRYPEX_AESCCM_Finish()` to get the authentication message. For CCM Encrypt/Decrypt API's, only `DataType = 8-bit` is supported by this version. The `HAL_CRYPEX_AESGCM_xxxx()` implementation is limited to 32bits inputs data length (Plain/Ciphertext, Header) compared with GCM standards specifications (800-38D).

6. Call HAL\_CRYPT\_DeInit() to deinitialize the CRYPT peripheral.

### 12.1.2 Extended AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES-128/192/256 using GCM and CCM chaining modes
- Decrypt cyphertext using AES-128/192/256 using GCM and CCM chaining modes
- Finish the processing. This function is available only for GCM and CCM

Three processing methods are available:

- Polling mode
- Interrupt mode
- DMA mode

This section contains the following APIs:

- [HAL\\_CRYPEx\\_AESCCM\\_Encrypt\(\)](#)
- [HAL\\_CRYPEx\\_AESGCM\\_Encrypt\(\)](#)
- [HAL\\_CRYPEx\\_AESGCM\\_Decrypt\(\)](#)
- [HAL\\_CRYPEx\\_AESGCM\\_Finish\(\)](#)
- [HAL\\_CRYPEx\\_AESCCM\\_Finish\(\)](#)
- [HAL\\_CRYPEx\\_AESCCM\\_Decrypt\(\)](#)
- [HAL\\_CRYPEx\\_AESGCM\\_Encrypt\\_IT\(\)](#)
- [HAL\\_CRYPEx\\_AESCCM\\_Encrypt\\_IT\(\)](#)
- [HAL\\_CRYPEx\\_AESGCM\\_Decrypt\\_IT\(\)](#)
- [HAL\\_CRYPEx\\_AESCCM\\_Decrypt\\_IT\(\)](#)
- [HAL\\_CRYPEx\\_AESGCM\\_Encrypt\\_DMA\(\)](#)
- [HAL\\_CRYPEx\\_AESCCM\\_Encrypt\\_DMA\(\)](#)
- [HAL\\_CRYPEx\\_AESGCM\\_Decrypt\\_DMA\(\)](#)
- [HAL\\_CRYPEx\\_AESCCM\\_Decrypt\\_DMA\(\)](#)

### 12.1.3 CRYPEx IRQ handler management

This section provides CRYPEx IRQ handler function.

This section contains the following APIs:

- [HAL\\_CRYPEx\\_GCMCCM\\_IRQHandler\(\)](#)

### 12.1.4 HAL\_CRYPEx\_AESCCM\_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYPEx_AESCCM_Encrypt(CRYPT_HandleTypeDef *hcrp, uint8_t *pPlainData, uint16_t Size, uint8_t *pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES CCM encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcrp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>



**12.1.5 HAL\_CRYPTEx\_AESGCM\_Encrypt**

Function Name	<b>HAL_StatusTypeDef HAL_CRYPTEx_AESGCM_Encrypt</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES GCM encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**12.1.6 HAL\_CRYPTEx\_AESGCM\_Decrypt**

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

**12.1.7 HAL\_CRYPTEx\_AESGCM\_Finish**

Function Name	<b>HAL_StatusTypeDef HAL_CRYPTEx_AESGCM_Finish</b> (CRYPT_HandleTypeDef * hcrypt, uint32_t Size, uint8_t * AuthTag, uint32_t Timeout)
Function Description	Computes the authentication TAG.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>Size</b>: Total length of the plain/cyphertext buffer</li> <li>• <b>AuthTag</b>: Pointer to the authentication buffer</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**12.1.8 HAL\_CRYPTEx\_AESCCM\_Finish**

Function Name	<b>HAL_StatusTypeDef HAL_CRYPTEx_AESCCM_Finish</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * AuthTag, uint32_t Timeout)
---------------	---

Function Description	Computes the authentication TAG for AES CCM mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>AuthTag</b>: Pointer to the authentication buffer</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>• This API is called after HAL_AES_CCM_Encrypt()/HAL_AES_CCM_Decrypt()</li> </ul>

### 12.1.9 HAL\_CRYPEX\_AESCCM\_Decrypt

Function Name	<b>HAL_StatusTypeDef HAL_CRYPEX_AESCCM_Decrypt (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)</b>
Function Description	Initializes the CRYPT peripheral in AES CCM decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 12.1.10 HAL\_CRYPEX\_AESGCM\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYPEX_AESGCM_Encrypt_IT (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYPT peripheral in AES GCM encryption mode using IT.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 12.1.11 HAL\_CRYPEX\_AESCCM\_Encrypt\_IT

Function Name	<b>HAL_StatusTypeDef HAL_CRYPEX_AESCCM_Encrypt_IT (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</b>
Function Description	Initializes the CRYPT peripheral in AES CCM encryption mode using interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pPlainData</b>: Pointer to the plaintext buffer</li> </ul>

- |               |   |
|---------------|---|
|               | <ul style="list-style-type: none"> <li>• <b>Size:</b> Length of the plaintext buffer, must be a multiple of 16</li> <li>• <b>pCypherData:</b> Pointer to the cyphertext buffer</li> </ul> |
| Return values | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>  |
- 12.1.12 HAL\_CRYPEX\_AESGCM\_Decrypt\_IT**
- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYPEX_AESGCM_Decrypt_IT</b><br>(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)   |
| Function Description | Initializes the CRYPT peripheral in AES GCM decryption mode using IT.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData:</b> Pointer to the cyphertext buffer</li> <li>• <b>Size:</b> Length of the cyphertext 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>  |
- 12.1.13 HAL\_CRYPEX\_AESCCM\_Decrypt\_IT**
- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYPEX_AESCCM_Decrypt_IT</b><br>(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)  |
| Function Description | Initializes the CRYPT peripheral in AES CCM decryption mode using interrupt then decrypted pCypherData.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData:</b> Pointer to the cyphertext 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>   |
- 12.1.14 HAL\_CRYPEX\_AESGCM\_Encrypt\_DMA**
- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_CRYPEX_AESGCM_Encrypt_DMA</b><br>(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)   |
| Function Description | Initializes the CRYPT peripheral in AES GCM encryption mode using DMA.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hcryp:</b> pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |
- 12.1.15 HAL\_CRYPEX\_AESCCM\_Encrypt\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_CRYPTEx_AESCCM_Encrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES CCM encryption mode using interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT 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 cyphertext buffer</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 12.1.16 HAL\_CRYPTEx\_AESGCM\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYPTEx_AESGCM_Decrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES GCM decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext buffer.</li> <li>• <b>Size</b>: Length of the cyphertext 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>

### 12.1.17 HAL\_CRYPTEx\_AESCCM\_Decrypt\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_CRYPTEx_AESCCM_Decrypt_DMA</b> (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES CCM decryption mode using DMA then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module</li> <li>• <b>pCypherData</b>: Pointer to the cyphertext 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>

### 12.1.18 HAL\_CRYPTEx\_GCMCCM\_IRQHandler

Function Name	<b>void HAL_CRYPTEx_GCMCCM_IRQHandler</b> (CRYPT_HandleTypeDef * hcrypt)
Function Description	This function handles CRYPTEx interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hcryp</b>: pointer to a CRYPTEx_HandleTypeDef structure that contains the configuration information for CRYPT module</li> </ul>

Return values

- None

## 12.2 CRYPEX Firmware driver defines

### 12.2.1 CRYPEX

#### **CRYP AlgoModeDirection**

CRYP\_CR\_ALGOMODE\_AES\_GCM\_ENCRYPT

CRYP\_CR\_ALGOMODE\_AES\_GCM\_DECRYPT

CRYP\_CR\_ALGOMODE\_AES\_CCM\_ENCRYPT

CRYP\_CR\_ALGOMODE\_AES\_CCM\_DECRYPT

#### **CRYP PhaseConfig**

CRYP\_PHASE\_INIT

CRYP\_PHASE\_HEADER

CRYP\_PHASE\_PAYLOAD

CRYP\_PHASE\_FINAL

#### **CRYP Exported Macros**

**\_\_HAL\_Cryp\_Set\_Phase** **Description:**

- Set the phase: Init, header, payload, final.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the CRYP handle.
- **\_\_PHASE\_\_**: The phase.

**Return value:**

- None

#### **CRYPEX\_Private\_define**

CRYPEX\_TIMEOUT\_VALUE

## 13 HAL DAC Generic Driver

### 13.1 DAC Firmware driver registers structures

#### 13.1.1 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

#### 13.1.2 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](#)

## 13.2 DAC Firmware driver API description

### 13.2.1 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 using HAL\_DACEx\_NoiseWaveGenerate()
2. Triangle wave using HAL\_DACEx\_TriangleWaveGenerate()

#### DAC data format

The DAC data format can be:

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

#### DAC data value to voltage 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,  $\text{DAC\_OUT1} = (3.3 * 868) / 4095 = 0.7\text{V}$

### 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

## 13.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL\_DAC\_Init()
- Configure DAC\_OUTx (DAC\_OUT1: PA4, DAC\_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL\_DAC\_ConfigChannel() function.
- Enable the DAC channel using HAL\_DAC\_Start() or HAL\_DAC\_Start\_DMA functions

### Polling mode IO operation

- Start the DAC peripheral using HAL\_DAC\_Start()
- To read the DAC last data output value, 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

### 13.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

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

This section contains the following APIs:

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

### 13.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.

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\\_DMAUnderrunCallbackCh1\(\)\*](#)

### 13.2.5 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\(\)\*](#)

### 13.2.6 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\(\)](#)

### 13.2.7 HAL\_DAC\_Init

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

### 13.2.8 HAL\_DAC\_DeInit

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

### 13.2.9 HAL\_DAC\_MspInit

Function Name	<b>void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)</b>
Function Description	Initializes the DAC MSP.
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>

### 13.2.10 HAL\_DAC\_MspDeInit

Function Name	<b>void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)</b>
Function Description	DeInitializes the DAC MSP.
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>

### 13.2.11 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>

### 13.2.12 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>

### 13.2.13 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>

### 13.2.14 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>
<b>13.2.15 HAL_DAC_GetValue</b>	
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>
<b>13.2.16 HAL_DAC_IRQHandler</b>	
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>
<b>13.2.17 HAL_DAC_ConvCpltCallbackCh1</b>	
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>
<b>13.2.18 HAL_DAC_ConvHalfCpltCallbackCh1</b>	
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>

### 13.2.19 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>

### 13.2.20 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>

### 13.2.21 HAL\_DAC\_ConfigChannel

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

### 13.2.22 HAL\_DAC\_SetValue

Function Name	<b>HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef *hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)</b>
Function Description	Set the specified data holding register value for DAC channel.
Parameters	<ul style="list-style-type: none"> <li><b>hdac:</b> pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.</li> <li><b>Channel:</b> The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2</li> </ul>

- selected
  - **Alignment:** 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
  - **Data:** Data to be loaded in the selected data holding register.
- Return values
- HAL status

### 13.2.23 HAL\_DAC\_GetState

- Function Name **HAL\_DAC\_StateTypeDef HAL\_DAC\_GetState (DAC\_HandleTypeDef \* hdac)**
- Function Description return the DAC state
- Parameters
- **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- Return values
- HAL state

### 13.2.24 HAL\_DAC\_GetError

- Function Name **uint32\_t HAL\_DAC\_GetError (DAC\_HandleTypeDef \* hdac)**
- Function Description Return the DAC error code.
- Parameters
- **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- Return values
- DAC Error Code

### 13.2.25 HAL\_DAC\_IRQHandler

- Function Name **void HAL\_DAC\_IRQHandler (DAC\_HandleTypeDef \* hdac)**
- Function Description Handles DAC interrupt request.
- Parameters
- **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- Return values
- None

### 13.2.26 HAL\_DAC\_ConvCpltCallbackCh1

- Function Name **void HAL\_DAC\_ConvCpltCallbackCh1 (DAC\_HandleTypeDef \* hdac)**
- Function Description Conversion complete callback in non blocking mode for Channel1.
- Parameters
- **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- Return values
- None

### 13.2.27 HAL\_DAC\_ConvHalfCpltCallbackCh1

- Function Name **void HAL\_DAC\_ConvHalfCpltCallbackCh1 (DAC\_HandleTypeDef \* hdac)**

Function Description	Conversion half DMA transfer callback in non blocking mode for Channel1.
Parameters	<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>

### 13.2.28 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>

### 13.2.29 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>

## 13.3 DAC Firmware driver defines

### 13.3.1 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\_DMAUNDERRUNCH1      DAC channel1 DAM underrun error

HAL\_DAC\_ERROR\_DMAUNDERRUNCH2      DAC channel2 DAM underrun error

HAL\_DAC\_ERROR\_DMA                      DMA error

#### **DAC Exported Macros**

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

`__HAL_DAC_ENABLE`

- 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

`__HAL_DAC_DISABLE`

`__HAL_DAC_ENABLE_IT`

`__HAL_DAC_DISABLE_IT`

`__HAL_DAC_GET_IT_SOURCE`



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)

**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

**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

`__HAL_DAC_GET_FLAG`

`__HAL_DAC_CLEAR_FLAG`

**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

## 14 HAL DAC Extension Driver

### 14.1 DACEx Firmware driver API description

#### 14.1.1 How to use this driver

- When Dual mode is enabled (i.e DAC Channel1 and Channel2 are used simultaneously) : Use HAL\_DACEx\_DualGetValue() to get digital data to be converted and use HAL\_DACEx\_DualSetValue() to set digital value to converted simultaneously in Channel 1 and Channel 2.
- Use HAL\_DACEx\_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL\_DACEx\_NoiseWaveGenerate() to generate Noise signal.

#### 14.1.2 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\(\)\*](#)

#### 14.1.3 HAL\_DACEx\_DualGetValue

Function Name	<b>uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)</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> </ul>
Return values	<ul style="list-style-type: none"> <li>• The selected DAC channel data output value.</li> </ul>

#### 14.1.4 HAL\_DACEx\_TriangleWaveGenerate

Function Name	<b>HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)</b>
Function Description	Enables or disables the selected DAC channel wave generation.

## Parameters

- **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC Channel1 selectedDAC\_CHANNEL\_2: DAC Channel2 selected
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:  
DAC\_TRIANGLEAMPLITUDE\_1: Select max triangle amplitude of 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

## Return values

- HAL status

### 14.1.5 HAL\_DACEx\_NoiseWaveGenerate

## Function Name

**HAL\_StatusTypeDef HAL\_DACEx\_NoiseWaveGenerate (DAC\_HandleTypeDef \*hdac, uint32\_t Channel, uint32\_t Amplitude)**

## Function Description

Enables or disables the selected DAC channel wave generation.

## Parameters

- **hdac:** pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values: DAC\_CHANNEL\_1: DAC Channel1 selectedDAC\_CHANNEL\_2: DAC Channel2 selected
- **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values: DAC\_LFSRUNMASK\_BIT0: Unmask DAC channel LFSR bit0 for noise wave 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  
 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

- HAL status

#### 14.1.6 HAL\_DACEx\_DualSetValue

**Function Name** `HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)`

**Function Description** Set the specified data holding register value for dual DAC channel.

**Parameters**

- **hdac**: pointer to a DAC\_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment**: Specifies the data alignment for dual channel DAC. This parameter can be one of the following values:  
 DAC\_ALIGN\_8B\_R: 8bit right data alignment selected  
 DAC\_ALIGN\_12B\_L: 12bit left data alignment selected  
 DAC\_ALIGN\_12B\_R: 12bit right data alignment selected
- **Data1**: Data for DAC Channel2 to be loaded in the selected data holding register.
- **Data2**: Data for DAC Channel1 to be loaded in the selected data holding register.

**Return values**

- HAL status

**Notes**

- In dual mode, a unique register access is required to write in both DAC channels at the same time.

#### 14.1.7 HAL\_DACEx\_ConvCpltCallbackCh2

**Function Name** `void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)`

**Function Description** Conversion complete callback in non blocking mode for Channel2.

**Parameters**

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

**Return values**

- None

#### 14.1.8 HAL\_DACEx\_ConvHalfCpltCallbackCh2

**Function Name** `void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)`

**Function Description** Conversion half DMA transfer callback in non blocking mode for

Channel2.

Parameters	<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>

### 14.1.9 HAL\_DACEx\_ErrorCallbackCh2

Function Name	<b>void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef *hdac)</b>
Function Description	Error DAC callback 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>

### 14.1.10 HAL\_DACEx\_DMAUnderrunCallbackCh2

Function Name	<b>void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef *hdac)</b>
Function Description	DMA underrun DAC callback 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>

## 14.2 DACEx Firmware driver defines

### 14.2.1 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

	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



## 15 HAL DCMI Generic Driver

### 15.1 DCMI Firmware driver registers structures

#### 15.1.1 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

### 15.2 DCMI Firmware driver API description

#### 15.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMI Interface. This sequence does not take into account the

configuration of the camera module, which should be made before to configure and enable the DCMi to capture images.

1. Program the required configuration through following parameters: horizontal and vertical polarity, pixel clock polarity, Capture Rate, Synchronization Mode, code of the frame delimiter and data width using HAL\_DCMi\_Init() function.
2. Configure the DMA2\_Stream1 channel1 to transfer Data from DCMi DR register to the destination memory buffer.
3. Program the required configuration through following parameters: DCMi mode, destination memory Buffer address and the data length and enable capture using HAL\_DCMi\_Start\_DMA() function.
4. Optionally, configure and Enable the CROP feature to select a rectangular window from the received image using HAL\_DCMi\_ConfigCrop() and HAL\_DCMi\_EnableCROP() functions
5. The capture can be stopped using HAL\_DCMi\_Stop() function.
6. To control DCMi state you can use the function HAL\_DCMi\_GetState().

### DCMi HAL driver macros list

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

- `__HAL_DCMi_ENABLE`: Enable the DCMi peripheral.
- `__HAL_DCMi_DISABLE`: Disable the DCMi peripheral.
- `__HAL_DCMi_GET_FLAG`: Get the DCMi pending flags.
- `__HAL_DCMi_CLEAR_FLAG`: Clear the DCMi pending flags.
- `__HAL_DCMi_ENABLE_IT`: Enable the specified DCMi interrupts.
- `__HAL_DCMi_DISABLE_IT`: Disable the specified DCMi interrupts.
- `__HAL_DCMi_GET_IT_SOURCE`: Check whether the specified DCMi interrupt has occurred or not.



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

## 15.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMi
- De-initialize the DCMi

This section contains the following APIs:

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

## 15.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length and Enables DCMi DMA request and enables DCMi capture
- Stop the DCMi capture.
- 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\(\)](#)

### 15.2.4 Peripheral Control functions

This section provides functions allowing to:

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

This section contains the following APIs:

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

### 15.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

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

This section contains the following APIs:

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

### 15.2.6 HAL\_DCMI\_Init

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

### 15.2.7 HAL\_DCMI\_DeInit

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

### 15.2.8 HAL\_DCMI\_MspInit



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

### 15.2.9 HAL\_DCMI\_MspDeInit

Function Name	<b>void HAL_DCMI_MspDeInit (DCMI_HandleTypeDef * hdcmi)</b>
Function Description	DeInitializes the DCMI MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b>: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 15.2.10 HAL\_DCMI\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_Start_DMA (DCMI_HandleTypeDef * hdcmi, uint32_t DCMI_Mode, uint32_t pData, uint32_t Length)</b>
Function Description	Enables DCMI DMA request and enables DCMI capture.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b>: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> <li>• <b>DCMI_Mode</b>: DCMI capture mode snapshot or continuous grab.</li> <li>• <b>pData</b>: The destination memory Buffer address (LCD Frame buffer).</li> <li>• <b>Length</b>: The length of capture to be transferred.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.11 HAL\_DCMI\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_DCMI_Stop (DCMI_HandleTypeDef * hdcmi)</b>
Function Description	Disable DCMI DMA request and Disable DCMI capture.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b>: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 15.2.12 HAL\_DCMI\_IRQHandler

Function Name	<b>void HAL_DCMI_IRQHandler (DCMI_HandleTypeDef * hdcmi)</b>
Function Description	Handles DCMI interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdcmi</b>: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for the DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**15.2.13 HAL\_DCMI\_ErrorCallback**

Function Name	<b>void HAL_DCMI_ErrorCallback (DCMI_HandleTypeDef * hdcmi)</b>
Function Description	Error DCMI callback.
Parameters	<ul style="list-style-type: none"> <li><b>hdcmi:</b> pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**15.2.14 HAL\_DCMI\_LineEventCallback**

Function Name	<b>void HAL_DCMI_LineEventCallback (DCMI_HandleTypeDef * hdcmi)</b>
Function Description	Line Event callback.
Parameters	<ul style="list-style-type: none"> <li><b>hdcmi:</b> pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**15.2.15 HAL\_DCMI\_VsyncEventCallback**

Function Name	<b>void HAL_DCMI_VsyncEventCallback (DCMI_HandleTypeDef * hdcmi)</b>
Function Description	VSYSNC Event callback.
Parameters	<ul style="list-style-type: none"> <li><b>hdcmi:</b> pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**15.2.16 HAL\_DCMI\_FrameEventCallback**

Function Name	<b>void HAL_DCMI_FrameEventCallback (DCMI_HandleTypeDef * hdcmi)</b>
Function Description	Frame Event callback.
Parameters	<ul style="list-style-type: none"> <li><b>hdcmi:</b> pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**15.2.17 HAL\_DCMI\_ConfigCROP**

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>YSize:</b> DCMI Line number</li> <li><b>XSize:</b> DCMI Pixel per line</li> <li><b>X0:</b> DCMI window X offset</li> </ul>

- **Y0:** DCMI window Y offset
- Return values
  - HAL status

### 15.2.18 HAL\_DCMI\_DisableCROP

- Function Name **HAL\_StatusTypeDef HAL\_DCMI\_DisableCROP (DCMI\_HandleTypeDef \* hdcmi)**
- Function Description Disable the Crop feature.
- Parameters
  - **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.
- Return values
  - HAL status

### 15.2.19 HAL\_DCMI\_EnableCROP

- Function Name **HAL\_StatusTypeDef HAL\_DCMI\_EnableCROP (DCMI\_HandleTypeDef \* hdcmi)**
- Function Description Enable the Crop feature.
- Parameters
  - **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.
- Return values
  - HAL status

### 15.2.20 HAL\_DCMI\_GetState

- Function Name **HAL\_DCMI\_StateTypeDef HAL\_DCMI\_GetState (DCMI\_HandleTypeDef \* hdcmi)**
- Function Description Return the DCMI state.
- Parameters
  - **hdcmi:** pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.
- Return values
  - HAL state

### 15.2.21 HAL\_DCMI\_GetError

- Function Name **uint32\_t HAL\_DCMI\_GetError (DCMI\_HandleTypeDef \* hdcmi)**
- Function Description Return the DCMI error code.
- Parameters
  - **hdcmi:** : pointer to a DCMI\_HandleTypeDef structure that contains the configuration information for DCMI.
- Return values
  - DCMI Error Code

## 15.3 DCMI Firmware driver defines

### 15.3.1 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	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset DCMI handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: specifies the DCMI handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_DCMI_ENABLE	<b>Description:</b> <ul style="list-style-type: none"> <li>Enable the DCMI.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: DCMI handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_DCMI_DISABLE	<b>Description:</b> <ul style="list-style-type: none"> <li>Disable the DCMI.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: DCMI handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_DCMI_GET_FLAG	<b>Description:</b> <ul style="list-style-type: none"> <li>Get the DCMI pending flags.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: DCMI handle</li> <li>__FLAG__: Get the specified flag. This parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>DCMI_FLAG_FRAMERI: Frame</li> </ul> </li> </ul>

### \_\_HAL\_DCMI\_CLEAR\_FLAG

- 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.

**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:**



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

- None

**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**

<b>DCMI_EXTEND_DATA_8B</b>	Interface captures 8-bit data on every pixel clock
<b>DCMI_EXTEND_DATA_10B</b>	Interface captures 10-bit data on every pixel clock
<b>DCMI_EXTEND_DATA_12B</b>	Interface captures 12-bit data on every pixel clock
<b>DCMI_EXTEND_DATA_14B</b>	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\_VSYNCR  
DCMI\_FLAG\_LINERI  
DCMI\_FLAG\_FRAMEMI  
DCMI\_FLAG\_OVFMI  
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/VSNC signals

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

**DCMi VSNC 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

## 16 HAL DCMi Extension Driver

### 16.1 DCMiEx Firmware driver registers structures

#### 16.1.1 DCMi\_CodesInitTypeDef

##### Data Fields

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

##### Field Documentation

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

#### 16.1.2 DCMi\_InitTypeDef

##### Data Fields

- *uint32\_t SynchroMode*
- *uint32\_t PCKPolarity*
- *uint32\_t VSPolarity*
- *uint32\_t HSPolarity*
- *uint32\_t CaptureRate*
- *uint32\_t ExtendedDataMode*
- *DCMi\_CodesInitTypeDef SyncroCode*
- *uint32\_t JPEGMode*
- *uint32\_t ByteSelectMode*
- *uint32\_t ByteSelectStart*
- *uint32\_t LineSelectMode*
- *uint32\_t LineSelectStart*

##### Field Documentation

- *uint32\_t DCMi\_InitTypeDef::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\*](#)
- ***uint32\_t DCMI\_InitTypeDef::ByteSelectMode***  
Specifies the data to be captured by the interface This parameter can be a value of [\*DCMIEx\\_Byte\\_Select\\_Mode\*](#)
- ***uint32\_t DCMI\_InitTypeDef::ByteSelectStart***  
Specifies if the data to be captured by the interface is even or odd This parameter can be a value of [\*DCMIEx\\_Byte\\_Select\\_Start\*](#)
- ***uint32\_t DCMI\_InitTypeDef::LineSelectMode***  
Specifies the line of data to be captured by the interface This parameter can be a value of [\*DCMIEx\\_Line\\_Select\\_Mode\*](#)
- ***uint32\_t DCMI\_InitTypeDef::LineSelectStart***  
Specifies if the line of data to be captured by the interface is even or odd This parameter can be a value of [\*DCMIEx\\_Line\\_Select\\_Start\*](#)

## 16.2 DCMIEx Firmware driver API description

### 16.2.1 DCMI peripheral extension features

Support of Black and White cameras

### 16.2.2 How to use this driver

This driver provides functions to manage the Black and White feature

### 16.2.3 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\(\)\*](#)

### 16.2.4 HAL\_DCMI\_Init



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

## 16.3 DCMIEx Firmware driver defines

### 16.3.1 DCMIEx

#### ***DCMIEx Byte Select Mode***

DCMI_BSM_ALL	Interface captures all received data
DCMI_BSM_OTHER	Interface captures every other byte from the received data
DCMI_BSM_ALTERNATE_4	Interface captures one byte out of four
DCMI_BSM_ALTERNATE_2	Interface captures two bytes out of four

#### ***DCMIEx Byte Select Start***

DCMI_OEBS_ODD	Interface captures first data from the frame/line start, second one being dropped
DCMI_OEBS_EVEN	Interface captures second data from the frame/line start, first one being dropped

#### ***DCMIEx Line Select Mode***

DCMI_LSM_ALL	Interface captures all received lines
DCMI_LSM_ALTERNATE_2	Interface captures one line out of two

#### ***DCMIEx Line Select Start***

DCMI_OELS_ODD	Interface captures first line from the frame start, second one being dropped
DCMI_OELS_EVEN	Interface captures second line from the frame start, first one being dropped

#### ***DCMIEx Private Macros***

IS\_DCMI\_BYTE\_SELECT\_MODE  
 IS\_DCMI\_BYTE\_SELECT\_START  
 IS\_DCMI\_LINE\_SELECT\_MODE  
 IS\_DCMI\_LINE\_SELECT\_START

## 17 HAL DMA2D Generic Driver

### 17.1 DMA2D Firmware driver registers structures

#### 17.1.1 DMA2D\_ColorTypeDef

##### Data Fields

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

##### Field Documentation

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

#### 17.1.2 DMA2D\_CLUTCfgTypeDef

##### Data Fields

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

##### Field Documentation

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

#### 17.1.3 DMA2D\_InitTypeDef

**Data Fields**

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

**Field Documentation**

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

**17.1.4 DMA2D\_LayerCfgTypeDef****Data Fields**

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

**Field Documentation**

- *uint32\_t DMA2D\_LayerCfgTypeDef::InputOffset*  
configures the DMA2D foreground offset. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0x3FFF.
- *uint32\_t DMA2D\_LayerCfgTypeDef::InputColorMode*  
configures the DMA2D foreground color mode . This parameter can be one value of [DMA2D\\_Input\\_Color\\_Mode](#)
- *uint32\_t DMA2D\_LayerCfgTypeDef::AlphaMode*  
configures the DMA2D foreground alpha mode. This parameter can be one value of [DMA2D\\_ALPHA\\_MODE](#)
- *uint32\_t DMA2D\_LayerCfgTypeDef::InputAlpha*  
Specifies the DMA2D foreground alpha value and color value in case of A8 or A4 color mode. This parameter must be a number between Min\_Data = 0x00000000 and Max\_Data = 0xFFFFFFFF in case of A8 or A4 color mode (ARGB). Otherwise, This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.

**17.1.5 \_\_DMA2D\_HandleTypeDef****Data Fields**

- *DMA2D\_TypeDef \* Instance*
- *DMA2D\_InitTypeDef Init*



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

#### Field Documentation

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

## 17.2 DMA2D Firmware driver API description

### 17.2.1 How to use this driver

1. Program the required configuration through following parameters: the Transfer Mode, the output color mode and the output offset using `HAL_DMA2D_Init()` function.
2. Program the required configuration through following parameters: the input color mode, the input color, input alpha value, alpha mode and the input offset using `HAL_DMA2D_ConfigLayer()` function for foreground or/and background layer.

#### Polling mode IO operation

- Configure the pdata, Destination and data length and Enable the transfer using `HAL_DMA2D_Start()`
- Wait for end of transfer using `HAL_DMA2D_PollForTransfer()`, at this stage user can specify the value of timeout according to his end application.

#### Interrupt mode IO operation

1. Configure the pdata, Destination and data length and Enable the transfer using HAL\_DMA2D\_Start\_IT()
2. Use HAL\_DMA2D\_IRQHandler() called under DMA2D\_IRQHandler() Interrupt subroutine
3. At the end of data transfer HAL\_DMA2D\_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e a member of DMA2D handle structure). In Register-to-Memory transfer mode, the pdata parameter is the register color, in Memory-to-memory or memory-to-memory with pixel format conversion the pdata is the source address. Configure the foreground source address, the background source address, the Destination and data length and Enable the transfer using HAL\_DMA2D\_BlendingStart() in polling mode and HAL\_DMA2D\_BlendingStart\_IT() in interrupt mode. HAL\_DMA2D\_BlendingStart() and HAL\_DMA2D\_BlendingStart\_IT() functions are used if the memory to memory with blending transfer mode is selected.
4. Optionally, configure and enable the CLUT using HAL\_DMA2D\_ConfigCLUT() HAL\_DMA2D\_EnableCLUT() functions.
5. Optionally, configure and enable LineInterrupt using the following function: HAL\_DMA2D\_ProgramLineEvent().
6. The transfer can be suspended, continued and aborted using the following functions: HAL\_DMA2D\_Suspend(), HAL\_DMA2D\_Resume(), HAL\_DMA2D\_Abort().
7. To control DMA2D state you can use the following function: HAL\_DMA2D\_GetState()

### DMA2D HAL driver macros list

Below the list of most used macros in DMA2D HAL driver :

- \_\_HAL\_DMA2D\_ENABLE: Enable the DMA2D peripheral.
- \_\_HAL\_DMA2D\_DISABLE: Disable the DMA2D peripheral.
- \_\_HAL\_DMA2D\_GET\_FLAG: Get the DMA2D pending flags.
- \_\_HAL\_DMA2D\_CLEAR\_FLAG: Clear the DMA2D pending flags.
- \_\_HAL\_DMA2D\_ENABLE\_IT: Enable the specified DMA2D interrupts.
- \_\_HAL\_DMA2D\_DISABLE\_IT: Disable the specified DMA2D interrupts.
- \_\_HAL\_DMA2D\_GET\_IT\_SOURCE: Check whether the specified DMA2D interrupt has occurred or not.



You can refer to the DMA2D HAL driver header file for more useful macros

## 17.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DMA2D
- De-initialize the DMA2D

This section contains the following APIs:

- [\*\*HAL\\_DMA2D\\_Init\(\)\*\*](#)
- [\*\*HAL\\_DMA2D\\_DeInit\(\)\*\*](#)
- [\*\*HAL\\_DMA2D\\_MspInit\(\)\*\*](#)
- [\*\*HAL\\_DMA2D\\_MspDeInit\(\)\*\*](#)

## 17.2.3 IO operation functions

This section provides functions allowing to:

- Configure the pdata, destination address and data size and Start DMA2D transfer.
- Configure the source for foreground and background, destination address and data size and Start MultiBuffer DMA2D transfer.
- Configure the pdata, destination address and data size and Start DMA2D transfer with interrupt.
- Configure the source for foreground and background, destination address and data size and Start MultiBuffer DMA2D transfer with interrupt.
- Abort DMA2D transfer.
- Suspend DMA2D transfer.
- Continue DMA2D transfer.
- Poll for transfer complete.
- handle DMA2D interrupt request.

This section contains the following APIs:

- [\*HAL\\_DMA2D\\_Start\(\)\*](#)
- [\*HAL\\_DMA2D\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_DMA2D\\_BlendingStart\(\)\*](#)
- [\*HAL\\_DMA2D\\_BlendingStart\\_IT\(\)\*](#)
- [\*HAL\\_DMA2D\\_Abort\(\)\*](#)
- [\*HAL\\_DMA2D\\_Suspend\(\)\*](#)
- [\*HAL\\_DMA2D\\_Resume\(\)\*](#)
- [\*HAL\\_DMA2D\\_PollForTransfer\(\)\*](#)
- [\*HAL\\_DMA2D\\_IRQHandler\(\)\*](#)

#### 17.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the DMA2D foreground or/and background parameters.
- Configure the DMA2D CLUT transfer.
- Enable DMA2D CLUT.
- Disable DMA2D CLUT.
- Configure the line watermark

This section contains the following APIs:

- [\*HAL\\_DMA2D\\_ConfigLayer\(\)\*](#)
- [\*HAL\\_DMA2D\\_ConfigCLUT\(\)\*](#)
- [\*HAL\\_DMA2D\\_EnableCLUT\(\)\*](#)
- [\*HAL\\_DMA2D\\_DisableCLUT\(\)\*](#)
- [\*HAL\\_DMA2D\\_ProgramLineEvent\(\)\*](#)

#### 17.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to :

- Check the DMA2D state
- Get error code

This section contains the following APIs:

- [\*HAL\\_DMA2D\\_GetState\(\)\*](#)
- [\*HAL\\_DMA2D\\_GetError\(\)\*](#)

#### 17.2.6 HAL\_DMA2D\_Init

Function Name

HAL\_StatusTypeDef HAL\_DMA2D\_Init

**(DMA2D\_HandleTypeDef \* hdma2d)**

Function Description	Initializes the DMA2D according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li><b>hdma2d:</b> pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**17.2.7 HAL\_DMA2D\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_DeInit (DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Deinitializes the DMA2D peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li><b>hdma2d:</b> pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**17.2.8 HAL\_DMA2D\_MspInit**

Function Name	<b>void HAL_DMA2D_MspInit (DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Initializes the DMA2D MSP.
Parameters	<ul style="list-style-type: none"> <li><b>hdma2d:</b> pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**17.2.9 HAL\_DMA2D\_MspDeInit**

Function Name	<b>void HAL_DMA2D_MspDeInit (DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	DeInitializes the DMA2D MSP.
Parameters	<ul style="list-style-type: none"> <li><b>hdma2d:</b> pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**17.2.10 HAL\_DMA2D\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Start (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)</b>
Function Description	Start the DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> <li><b>hdma2d:</b> pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li><b>pdata:</b> Configure the source memory Buffer address if the memory to memory or memory to memory with pixel format conversion DMA2D mode is selected, and configure the color value if register to memory DMA2D mode is selected.</li> </ul>

- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination.
- **Height:** The height of data to be transferred from source to destination.

Return values

- HAL status

### 17.2.11 HAL\_DMA2D\_Start\_IT

Function Name

**HAL\_StatusTypeDef HAL\_DMA2D\_Start\_IT**  
**(DMA2D\_HandleTypeDef \* hdma2d, uint32\_t pdata, uint32\_t DstAddress, uint32\_t Width, uint32\_t Height)**

Function Description

Start the DMA2D Transfer with interrupt enabled.

Parameters

- **hdma2d:** pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata:** Configure the source memory Buffer address if the memory to memory or memory to memory with pixel format conversion DMA2D mode is selected, and configure the color value if register to memory DMA2D mode is selected.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination.
- **Height:** The height of data to be transferred from source to destination.

Return values

- HAL status

### 17.2.12 HAL\_DMA2D\_BlendingStart

Function Name

**HAL\_StatusTypeDef HAL\_DMA2D\_BlendingStart**  
**(DMA2D\_HandleTypeDef \* hdma2d, uint32\_t SrcAddress1, uint32\_t SrcAddress2, uint32\_t DstAddress, uint32\_t Width, uint32\_t Height)**

Function Description

Start the multi-source DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1:** The source memory Buffer address of the foreground layer.
- **SrcAddress2:** The source memory Buffer address of the background layer.
- **DstAddress:** The destination memory Buffer address
- **Width:** The width of data to be transferred from source to destination.
- **Height:** The height of data to be transferred from source to destination.

Return values

- HAL status

### 17.2.13 HAL\_DMA2D\_BlendingStart\_IT

Function Name

**HAL\_StatusTypeDef HAL\_DMA2D\_BlendingStart\_IT**  
**(DMA2D\_HandleTypeDef \* hdma2d, uint32\_t SrcAddress1, uint32\_t SrcAddress2, uint32\_t DstAddress, uint32\_t Width,**

uint32\_t Height)

Function Description	Start the multi-source DMA2D Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>SrcAddress1</b>: The source memory Buffer address of the foreground layer.</li> <li>• <b>SrcAddress2</b>: The source memory Buffer address of the background layer.</li> <li>• <b>DstAddress</b>: The destination memory Buffer address.</li> <li>• <b>Width</b>: The width of data to be transferred from source to destination.</li> <li>• <b>Height</b>: The height of data to be transferred from source to destination.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 17.2.14 HAL\_DMA2D\_Abort

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Abort</b> (DMA2D_HandleTypeDef * hdma2d)
Function Description	Abort the DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: : pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 17.2.15 HAL\_DMA2D\_Suspend

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Suspend</b> (DMA2D_HandleTypeDef * hdma2d)
Function Description	Suspend the DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 17.2.16 HAL\_DMA2D\_Resume

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_Resume</b> (DMA2D_HandleTypeDef * hdma2d)
Function Description	Resume the DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 17.2.17 HAL\_DMA2D\_PollForTransfer

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_PollForTransfer</b> (DMA2D_HandleTypeDef * hdma2d, uint32_t Timeout)
Function Description	Polling for transfer complete or CLUT loading.

Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 17.2.18 HAL\_DMA2D\_IRQHandler

Function Name	<b>void HAL_DMA2D_IRQHandler (DMA2D_HandleTypeDef * hdma2d)</b>
Function Description	Handles DMA2D interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 17.2.19 HAL\_DMA2D\_ConfigLayer

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_ConfigLayer (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)</b>
Function Description	Configure the DMA2D Layer according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>LayerIdx</b>: DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 17.2.20 HAL\_DMA2D\_ConfigCLUT

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_ConfigCLUT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)</b>
Function Description	Configure the DMA2D CLUT Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> <li>• <b>CLUTCfg</b>: pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.</li> <li>• <b>LayerIdx</b>: DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 17.2.21 HAL\_DMA2D\_EnableCLUT

Function Name	<b>HAL_StatusTypeDef HAL_DMA2D_EnableCLUT (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)</b>
Function Description	Enable the DMA2D CLUT Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hdma2d</b>: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.</li> </ul>

- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- HAL status

### 17.2.22 HAL\_DMA2D\_DisableCLUT

**Function Name** `HAL_StatusTypeDef HAL_DMA2D_DisableCLUT(DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)`

**Function Description** Disable the DMA2D CLUT Transfer.

- Parameters**
- **hdma2d:** pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
  - **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- HAL status

### 17.2.23 HAL\_DMA2D\_ProgramLineEvent

**Function Name** `HAL_StatusTypeDef HAL_DMA2D_ProgramLineEvent(DMA2D_HandleTypeDef * hdma2d, uint32_t Line)`

**Function Description** Define the configuration of the line watermark .

- Parameters**
- **hdma2d:** pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.
  - **Line:** Line Watermark configuration.

Return values

- HAL status

### 17.2.24 HAL\_DMA2D\_GetState

**Function Name** `HAL_DMA2D_StateTypeDef HAL_DMA2D_GetState(DMA2D_HandleTypeDef * hdma2d)`

**Function Description** Return the DMA2D state.

- Parameters**
- **hdma2d:** pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- HAL state

### 17.2.25 HAL\_DMA2D\_GetError

**Function Name** `uint32_t HAL_DMA2D_GetError (DMA2D_HandleTypeDef * hdma2d)`

**Function Description** Return the DMA2D error code.

- Parameters**
- **hdma2d:** : pointer to a DMA2D\_HandleTypeDef structure that contains the configuration information for DMA2D.

Return values

- DMA2D Error Code



## 17.3 DMA2D Firmware driver defines

### 17.3.1 DMA2D

#### ***DMA2D ALPHA MODE***

DMA2D_NO_MODIF_ALPHA	No modification of the alpha channel value
DMA2D_REPLACE_ALPHA	Replace original alpha channel value by programmed alpha value
DMA2D_COMBINE_ALPHA	Replace original alpha channel value by programmed alpha value with original alpha channel value

#### ***DMA2D CLUT CM***

DMA2D_CCM_ARGB8888	ARGB8888 DMA2D C-LUT color mode
DMA2D_CCM_RGB888	RGB888 DMA2D C-LUT color mode

#### ***DMA2D Color Mode***

DMA2D_ARGB8888	ARGB8888 DMA2D color mode
DMA2D_RGB888	RGB888 DMA2D color mode
DMA2D_RGB565	RGB565 DMA2D color mode
DMA2D_ARGB1555	ARGB1555 DMA2D color mode
DMA2D_ARGB4444	ARGB4444 DMA2D color mode

#### ***DMA2D COLOR VALUE***

COLOR_VALUE	color value mask
-------------	------------------

#### ***DMA2D DeadTime***

LINE\_WATERMARK

#### ***DMA2D Error Code***

HAL_DMA2D_ERROR_NONE	No error
HAL_DMA2D_ERROR_TE	Transfer error
HAL_DMA2D_ERROR_CE	Configuration error
HAL_DMA2D_ERROR_TIMEOUT	Timeout error

#### ***DMA2D Exported Macros***

__HAL_DMA2D_RESET_HANDLE_STATE	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset DMA2D handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: specifies the DMA2D handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_DMA2D_ENABLE	<b>Description:</b> <ul style="list-style-type: none"> <li>Enable the DMA2D.</li> </ul> <b>Parameters:</b>

`__HAL_DMA2D_DISABLE`

- `__HANDLE__`: DMA2D handle

**Return value:**

- None.

**Description:**

- Disable the DMA2D.

**Parameters:**

- `__HANDLE__`: DMA2D handle

**Return value:**

- None.

**Description:**

- Get the DMA2D pending flags.

**Parameters:**

- `__HANDLE__`: DMA2D handle
- `__FLAG__`: Get the specified flag. This parameter can be any combination of the following values:
  - `DMA2D_FLAG_CE`: Configuration error flag
  - `DMA2D_FLAG CTC`: C-LUT transfer complete flag
  - `DMA2D_FLAG CAE`: C-LUT access error flag
  - `DMA2D_FLAG_TW`: Transfer Watermark flag
  - `DMA2D_FLAG TC`: Transfer complete flag
  - `DMA2D_FLAG TE`: Transfer error flag

**Return value:**

- The: state of FLAG.

**Description:**

- Clears the DMA2D pending flags.

**Parameters:**

- `__HANDLE__`: DMA2D handle
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
  - `DMA2D_FLAG_CE`: Configuration error flag
  - `DMA2D_FLAG CTC`: C-LUT transfer complete flag
  - `DMA2D_FLAG CAE`: C-LUT access error flag
  - `DMA2D_FLAG_TW`: Transfer Watermark flag

`__HAL_DMA2D_GET_FLAG``__HAL_DMA2D_CLEAR_FLAG`

- DMA2D\_FLAG\_TC: Transfer complete flag
- DMA2D\_FLAG\_TE: Transfer error flag

**Return value:**

- None

**Description:**

- Enables the specified DMA2D interrupts.

**Parameters:**

- \_\_HANDLE\_\_: DMA2D handle
- \_\_INTERRUPT\_\_: specifies the DMA2D interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DMA2D\_IT\_CE: Configuration error interrupt mask
  - DMA2D\_IT CTC: C-LUT transfer complete interrupt mask
  - DMA2D\_IT CAE: C-LUT access error interrupt mask
  - DMA2D\_IT\_TW: Transfer Watermark interrupt mask
  - DMA2D\_IT\_TC: Transfer complete interrupt mask
  - DMA2D\_IT\_TE: Transfer error interrupt mask

**Return value:**

- None

**Description:**

- Disables the specified DMA2D interrupts.

**Parameters:**

- \_\_HANDLE\_\_: DMA2D handle
- \_\_INTERRUPT\_\_: specifies the DMA2D interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DMA2D\_IT\_CE: Configuration error interrupt mask
  - DMA2D\_IT CTC: C-LUT transfer complete interrupt mask
  - DMA2D\_IT CAE: C-LUT access error interrupt mask
  - DMA2D\_IT\_TW: Transfer Watermark interrupt mask
  - DMA2D\_IT\_TC: Transfer complete interrupt mask

\_\_HAL\_DMA2D\_ENABLE\_IT

\_\_HAL\_DMA2D\_DISABLE\_IT

- DMA2D\_IT\_TE: Transfer error interrupt mask

**Return value:**

- None

**Description:**

- Checks whether the specified DMA2D interrupt has occurred or not.

**Parameters:**

- \_\_HANDLE\_\_: DMA2D handle
- \_\_INTERRUPT\_\_: specifies the DMA2D interrupt source to check. This parameter can be one of the following values:
  - DMA2D\_IT\_CE: Configuration error interrupt mask
  - DMA2D\_IT\_CTC: C-LUT transfer complete interrupt mask
  - DMA2D\_IT\_CAE: C-LUT access error interrupt mask
  - DMA2D\_IT\_TW: Transfer Watermark interrupt mask
  - DMA2D\_IT\_TC: Transfer complete interrupt mask
  - DMA2D\_IT\_TE: Transfer error interrupt mask

**Return value:**

- The: state of INTERRUPT.

\_\_HAL\_DMA2D\_GET\_IT\_SOURCE

**DMA2D Exported Types**

MAX\_DMA2D\_LAYER

**DMA2D Flag**

DMA2D_FLAG_CE	Configuration Error Interrupt Flag
DMA2D_FLAG_CTC	C-LUT Transfer Complete Interrupt Flag
DMA2D_FLAG_CAE	C-LUT Access Error Interrupt Flag
DMA2D_FLAG_TW	Transfer Watermark Interrupt Flag
DMA2D_FLAG_TC	Transfer Complete Interrupt Flag
DMA2D_FLAG_TE	Transfer Error Interrupt Flag

**DMA2D Input Color Mode**

CM_ARGB8888	ARGB8888 color mode
CM_RGB888	RGB888 color mode
CM_RGB565	RGB565 color mode
CM_ARGB1555	ARGB1555 color mode
CM_ARGB4444	ARGB4444 color mode

CM_L8	L8 color mode
CM_AL44	AL44 color mode
CM_AL88	AL88 color mode
CM_L4	L4 color mode
CM_A8	A8 color mode
CM_A4	A4 color mode

**DMA2D Interrupts**

DMA2D_IT_CE	Configuration Error Interrupt
DMA2D_IT_CTC	C-LUT Transfer Complete Interrupt
DMA2D_IT_CAE	C-LUT Access Error Interrupt
DMA2D_IT_TW	Transfer Watermark Interrupt
DMA2D_IT_TC	Transfer Complete Interrupt
DMA2D_IT_TE	Transfer Error Interrupt

**DMA2D Mode**

DMA2D_M2M	DMA2D memory to memory transfer mode
DMA2D_M2M_PFC	DMA2D memory to memory with pixel format conversion transfer mode
DMA2D_M2M_BLEND	DMA2D memory to memory with blending transfer mode
DMA2D_R2M	DMA2D register to memory transfer mode

**DMA2D Offset**

DMA2D_OFFSET	Line Offset
--------------	-------------

**DMA2D Private Defines**

HAL\_TIMEOUT\_DMA2D\_ABORT  
HAL\_TIMEOUT\_DMA2D\_SUSPEND

**DMA2D Private Macros**

IS\_DMA2D\_LAYER  
IS\_DMA2D\_MODE  
IS\_DMA2D\_CMODE  
IS\_DMA2D\_COLOR  
IS\_DMA2D\_LINE  
IS\_DMA2D\_PIXEL  
IS\_DMA2D\_OFFSET  
IS\_DMA2D\_INPUT\_COLOR\_MODE  
IS\_DMA2D\_ALPHA\_MODE  
IS\_DMA2D\_CLUT\_CM  
IS\_DMA2D\_CLUT\_SIZE  
IS\_DMA2D\_LineWatermark

---

IS\_DMA2D\_IT

IS\_DMA2D\_GET\_FLAG

**DMA2D SIZE**

DMA2D\_PIXEL            DMA2D pixel per line

DMA2D\_LINE            DMA2D number of line

**DMA2D Size Clut**

DMA2D\_CLUT\_SIZE    DMA2D C-LUT size

## 18 HAL DMA Generic Driver

### 18.1 DMA Firmware driver registers structures

#### 18.1.1 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.

### 18.1.2 `__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`**

#### 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

## 18.2 DMA Firmware driver API description

### 18.2.1 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

## 18.2.2 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\_DeInit\(\)`](#)

## 18.2.3 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\(\)`](#)

## 18.2.4 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\(\)](#)

## 18.2.5 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>

## 18.2.6 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>

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

## 18.2.8 HAL\_DMA\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t</b>
---------------	--

	<b>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>
<b>18.2.9 HAL_DMA_Abort</b>	
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>
<b>18.2.10 HAL_DMA_PollForTransfer</b>	
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>
<b>18.2.11 HAL_DMA_IRQHandler</b>	
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>

**18.2.12 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>

**18.2.13 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>

**18.3 DMA Firmware driver defines****18.3.1 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\_HTIF0\_4

DMA\_FLAG\_TCIF0\_4

DMA\_FLAG\_FEIF1\_5

DMA\_FLAG\_DMEIF1\_5

DMA\_FLAG\_TEIF1\_5

DMA\_FLAG\_HTIF1\_5

DMA\_FLAG\_TCIF1\_5

DMA\_FLAG\_FEIF2\_6

DMA\_FLAG\_DMEIF2\_6

DMA\_FLAG\_TEIF2\_6

DMA\_FLAG\_HTIF2\_6

DMA\_FLAG\_TCIF2\_6

DMA\_FLAG\_FEIF3\_7

DMA\_FLAG\_DMEIF3\_7

DMA\_FLAG\_TEIF3\_7

DMA\_FLAG\_HTIF3\_7

DMA\_FLAG\_TCIF3\_7

**DMA 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 requestsTIM\_DMA\_ID\_CC2    Index of the DMA handle used for Capture/Compare 2  
DMA requestsTIM\_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



## 19 HAL DMA Extension Driver

### 19.1 DMAEx Firmware driver API description

#### 19.1.1 How to use this driver

The DMA Extension HAL driver can be used as follows:

- Start a multi buffer transfer using the `HAL_DMA_MultiBufferStart()` function for polling mode or `HAL_DMA_MultiBufferStart_IT()` for interrupt mode. 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.

#### 19.1.2 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\(\)`](#)

#### 19.1.3 HAL\_DMAEx\_MultiBufferStart

Function Name	<b>HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart</b> (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)
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>

#### 19.1.4 HAL\_DMAEx\_MultiBufferStart\_IT

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

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

**Function Description** Starts the multi\_buffer DMA Transfer with interrupt enabled.

**Parameters**

- **hdma:** pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **SecondMemAddress:** The second memory Buffer address in case of multi buffer Transfer
- **DataLength:** The length of data to be transferred from source to destination

**Return values**

- HAL status

### 19.1.5 HAL\_DMAEx\_ChangeMemory

**Function Name** HAL\_StatusTypeDef HAL\_DMAEx\_ChangeMemory (DMA\_HandleTypeDef \* hdma, uint32\_t Address, HAL\_DMA\_MemoryTypeDef memory)

**Function Description** Change the memory0 or memory1 address on the fly.

**Parameters**

- **hdma:** pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **Address:** The new address
- **memory:** the memory to be changed, This parameter can be one of the following values: MEMORY0 / MEMORY1

**Return values**

- HAL status

**Notes**

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

## 20 HAL ETH Generic Driver

### 20.1 ETH Firmware driver registers structures

#### 20.1.1 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 MediaInterface*

##### Field Documentation

- *uint32\_t ETH\_InitTypeDef::AutoNegotiation*  
Selects or not the AutoNegotiation mode for the external PHY The AutoNegotiation allows an automatic setting of the Speed (10/100Mbps) and the mode (half/full-duplex). This parameter can be a value of [ETH\\_AutoNegotiation](#)
- *uint32\_t ETH\_InitTypeDef::Speed*  
Sets the Ethernet speed: 10/100 Mbps. This parameter can be a value of [ETH\\_Speed](#)
- *uint32\_t ETH\_InitTypeDef::DuplexMode*  
Selects the MAC duplex mode: Half-Duplex or Full-Duplex mode This parameter can be a value of [ETH\\_Duplex\\_Mode](#)
- *uint16\_t ETH\_InitTypeDef::PhyAddress*  
Ethernet PHY address. This parameter must be a number between Min\_Data = 0 and Max\_Data = 32
- *uint8\_t\* ETH\_InitTypeDef::MACAddr*  
MAC Address of used Hardware: must be pointer on an array of 6 bytes
- *uint32\_t ETH\_InitTypeDef::RxMode*  
Selects the Ethernet Rx mode: Polling mode, Interrupt mode. This parameter can be a value of [ETH\\_Rx\\_Mode](#)
- *uint32\_t ETH\_InitTypeDef::ChecksumMode*  
Selects if the checksum is check by hardware or by software. This parameter can be a value of [ETH\\_Checksum\\_Mode](#)
- *uint32\_t ETH\_InitTypeDef::MediaInterface*  
Selects the media-independent interface or the reduced media-independent interface. This parameter can be a value of [ETH\\_Media\\_Interface](#)

#### 20.1.2 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\\_Unicast\\_Frames\\_Filter](#)
- **`uint32_t ETH_MACInitTypeDef::HashTableHigh`**  
This field holds the higher 32 bits of Hash table. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFFFFF
- **`uint32_t ETH_MACInitTypeDef::HashTableLow`**  
This field holds the lower 32 bits of Hash table. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFFFFFF
- **`uint32_t ETH_MACInitTypeDef::PauseTime`**  
This field holds the value to be used in the Pause Time field in the transmit control frame. This parameter must be a number between Min\_Data = 0x0 and Max\_Data = 0xFFFF
- **`uint32_t ETH_MACInitTypeDef::ZeroQuantaPause`**  
Selects or not the automatic generation of Zero-Quanta Pause Control frames. This parameter can be a value of [ETH\\_Zero\\_Quanta\\_Pause](#)

- ***uint32\_t ETH\_MACInitTypeDef::PauseLowThreshold***  
This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Frame. This parameter can be a value of [ETH\\_Pause\\_Low\\_Threshold](#)
- ***uint32\_t ETH\_MACInitTypeDef::UnicastPauseFrameDetect***  
Selects or not the MAC detection of the Pause frames (with MAC Address0 unicast address and unique multicast address). This parameter can be a value of [ETH\\_Unicast\\_Pause\\_Frame\\_Detect](#)
- ***uint32\_t ETH\_MACInitTypeDef::ReceiveFlowControl***  
Enables or disables the MAC to decode the received Pause frame and disable its transmitter for a specified time (Pause Time) This parameter can be a value of [ETH\\_Receive\\_Flow\\_Control](#)
- ***uint32\_t ETH\_MACInitTypeDef::TransmitFlowControl***  
Enables or disables the MAC to transmit Pause frames (Full-Duplex mode) or the MAC back-pressure operation (Half-Duplex mode) This parameter can be a value of [ETH\\_Transmit\\_Flow\\_Control](#)
- ***uint32\_t ETH\_MACInitTypeDef::VLANTagComparison***  
Selects the 12-bit VLAN identifier or the complete 16-bit VLAN tag for comparison and filtering. This parameter can be a value of [ETH\\_VLAN\\_Tag\\_Comparison](#)
- ***uint32\_t ETH\_MACInitTypeDef::VLANTagIdentifier***  
Holds the VLAN tag identifier for receive frames

### 20.1.3 ETH\_DMAInitTypeDef

#### 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\_DMAInitTypeDef::DropTCPIPChecksumErrorFrame***  
Selects or not the Dropping of TCP/IP Checksum Error Frames. This parameter can be a value of [ETH\\_Drop\\_TCP\\_IP\\_Checksum\\_Error\\_Frame](#)
- ***uint32\_t ETH\_DMAInitTypeDef::ReceiveStoreForward***  
Enables or disables the Receive store and forward mode. This parameter can be a value of [ETH\\_Receive\\_Store\\_Forward](#)

- **`uint32_t ETH_DMAInitTypeDef::FlushReceivedFrame`**  
Enables or disables the flushing of received frames. This parameter can be a value of [`ETH\_Flush\_Received\_Frame`](#)
- **`uint32_t ETH_DMAInitTypeDef::TransmitStoreForward`**  
Enables or disables Transmit store and forward mode. This parameter can be a value of [`ETH\_Transmit\_Store\_Forward`](#)
- **`uint32_t ETH_DMAInitTypeDef::TransmitThresholdControl`**  
Selects or not the Transmit Threshold Control. This parameter can be a value of [`ETH\_Transmit\_Threshold\_Control`](#)
- **`uint32_t ETH_DMAInitTypeDef::ForwardErrorFrames`**  
Selects or not the forward to the DMA of erroneous frames. This parameter can be a value of [`ETH\_Forward\_Error\_Frames`](#)
- **`uint32_t ETH_DMAInitTypeDef::ForwardUndersizedGoodFrames`**  
Enables or disables the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC) This parameter can be a value of [`ETH\_Forward\_Undersized\_Good\_Frames`](#)
- **`uint32_t ETH_DMAInitTypeDef::ReceiveThresholdControl`**  
Selects the threshold level of the Receive FIFO. This parameter can be a value of [`ETH\_Receive\_Threshold\_Control`](#)
- **`uint32_t ETH_DMAInitTypeDef::SecondFrameOperate`**  
Selects or not the Operate on second frame mode, which allows the DMA to process a second frame of Transmit data even before obtaining the status for the first frame. This parameter can be a value of [`ETH\_Second\_Frame\_Operate`](#)
- **`uint32_t ETH_DMAInitTypeDef::AddressAlignedBeats`**  
Enables or disables the Address Aligned Beats. This parameter can be a value of [`ETH\_Address\_Aligned\_Beats`](#)
- **`uint32_t ETH_DMAInitTypeDef::FixedBurst`**  
Enables or disables the AHB Master interface fixed burst transfers. This parameter can be a value of [`ETH\_Fixed\_Burst`](#)
- **`uint32_t ETH_DMAInitTypeDef::RxDMABurstLength`**  
Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of [`ETH\_Rx\_DMA\_Burst\_Length`](#)
- **`uint32_t ETH_DMAInitTypeDef::TxDMABurstLength`**  
Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of [`ETH\_Tx\_DMA\_Burst\_Length`](#)
- **`uint32_t ETH_DMAInitTypeDef::EnhancedDescriptorFormat`**  
Enables the enhanced descriptor format. This parameter can be a value of [`ETH\_DMA\_Enhanced\_descriptor\_format`](#)
- **`uint32_t ETH_DMAInitTypeDef::DescriptorSkipLength`**  
Specifies the number of word to skip between two unchained descriptors (Ring mode) This parameter must be a number between Min\_Data = 0 and Max\_Data = 32
- **`uint32_t ETH_DMAInitTypeDef::DMAArbitration`**  
Selects the DMA Tx/Rx arbitration. This parameter can be a value of [`ETH\_DMA\_Arbitration`](#)

## 20.1.4 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

### 20.1.5 ETH\_DMARxFramelInfos

#### Data Fields

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

#### Field Documentation

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



## 20.1.6 ETH\_HandleTypeDef

### Data Fields

- **ETH\_TypeDef \* Instance**
- **ETH\_InitTypeDef Init**
- **uint32\_t LinkStatus**
- **ETH\_DMADescTypeDef \* RxDesc**
- **ETH\_DMADescTypeDef \* TxDesc**
- **ETH\_DMARxFramInfos 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\_DMARxFramInfos ETH\_HandleTypeDef::RxFrameInfos**  
last Rx frame infos
- **\_\_IO HAL\_ETH\_StateTypeDef ETH\_HandleTypeDef::State**  
ETH communication state
- **HAL\_LockTypeDef ETH\_HandleTypeDef::Lock**  
ETH Lock

## 20.2 ETH Firmware driver API description

### 20.2.1 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.

### 20.2.2 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\\_DeInit\(\)\*](#)
- [\*HAL\\_ETH\\_DMATxDescListInit\(\)\*](#)
- [\*HAL\\_ETH\\_DMARxDescListInit\(\)\*](#)
- [\*HAL\\_ETH\\_MspInit\(\)\*](#)
- [\*HAL\\_ETH\\_MspDeInit\(\)\*](#)

### 20.2.3 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\(\)\*](#)

## 20.2.4 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\(\)`](#)

## 20.2.5 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\(\)`](#)

## 20.2.6 HAL\_ETH\_Init

Function Name	<b>HAL_StatusTypeDef HAL_ETH_Init (ETH_HandleTypeDef * heth)</b>
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>

## 20.2.7 HAL\_ETH\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_ETH_DeInit (ETH_HandleTypeDef * heth)</b>
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>

## 20.2.8 HAL\_ETH\_DMATxDescListInit

Function Name	<b>HAL_StatusTypeDef HAL_ETH_DMATxDescListInit (ETH_HandleTypeDef * heth, ETH_DMADescTypeDef * DMATxDescTab, uint8_t * TxBuff, uint32_t TxBuffCount)</b>
Function Description	Initializes the DMA Tx descriptors in chain mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>heth:</b> pointer to a <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
- Return values
- HAL status

### 20.2.9 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

### 20.2.10 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

### 20.2.11 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

### 20.2.12 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

### 20.2.13 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 <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>

#### 20.2.14 HAL\_ETH\_GetReceivedFrame\_IT

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 <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>

#### 20.2.15 HAL\_ETH\_IRQHandler

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 <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>

#### 20.2.16 HAL\_ETH\_TxCpltCallback

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 <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 20.2.17 HAL\_ETH\_RxCpltCallback

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 <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 20.2.18 HAL\_ETH\_ErrorCallback

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 <code>ETH_HandleTypeDef</code> structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

## 20.2.19 HAL\_ETH\_ReadPHYRegister

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ReadPHYRegister</b> (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t * RegValue)
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>

## 20.2.20 HAL\_ETH\_WritePHYRegister

Function Name	<b>HAL_StatusTypeDef HAL_ETH_WritePHYRegister</b> (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t RegValue)
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>

## 20.2.21 HAL\_ETH\_Start

Function Name	<b>HAL_StatusTypeDef HAL_ETH_Start</b> (ETH_HandleTypeDef * heth)
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>

## 20.2.22 HAL\_ETH\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_ETH_Stop</b> (ETH_HandleTypeDef * heth)
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>

**20.2.23 HAL\_ETH\_ConfigMAC**

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ConfigMAC</b> (ETH_HandleTypeDef * heth, ETH_MACInitTypeDef * macconf)
Function Description	Set ETH MAC Configuration.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li><b>macconf</b>: MAC Configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**20.2.24 HAL\_ETH\_ConfigDMA**

Function Name	<b>HAL_StatusTypeDef HAL_ETH_ConfigDMA</b> (ETH_HandleTypeDef * heth, ETH_DMAInitTypeDef * dmaconf)
Function Description	Sets ETH DMA Configuration.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> <li><b>dmaconf</b>: DMA Configuration structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**20.2.25 HAL\_ETH\_GetState**

Function Name	<b>HAL_ETH_StateTypeDef HAL_ETH_GetState</b> (ETH_HandleTypeDef * heth)
Function Description	Return the ETH HAL state.
Parameters	<ul style="list-style-type: none"> <li><b>heth</b>: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

**20.3 ETH Firmware driver defines****20.3.1 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\_BACKOFFLIMIT\_8

ETH\_BACKOFFLIMIT\_4

ETH\_BACKOFFLIMIT\_1

***ETH Broadcast Frames Reception***

ETH\_BROADCASTFRAMESRECEPTION\_ENABLE

ETH\_BROADCASTFRAMESRECEPTION\_DISABLE

***ETH Buffers setting***

ETH\_MAX\_PACKET\_SIZE                      ETH\_HEADER + ETH\_EXTRA + ETH\_VLAN\_TAG +  
ETH\_MAX\_ETH\_PAYLOAD + ETH\_CRC

ETH\_HEADER                                  6 byte Dest addr, 6 byte Src addr, 2 byte length/type

ETH\_CRC                                      Ethernet CRC

ETH\_EXTRA                                   Extra bytes in some cases

ETH\_VLAN\_TAG                               optional 802.1q VLAN Tag

ETH\_MIN\_ETH\_PAYLOAD                      Minimum Ethernet payload size

ETH\_MAX\_ETH\_PAYLOAD                      Maximum Ethernet payload size

ETH\_JUMBO\_FRAME\_PAYLOAD                Jumbo frame payload size

ETH\_RX\_BUF\_SIZE

ETH\_RXBUFNB

ETH\_TX\_BUF\_SIZE

ETH\_TXBUFNB

***ETH Carrier Sense***

ETH\_CARRIERSENCE\_ENABLE

ETH\_CARRIERSENCE\_DISABLE

***ETH Checksum Mode***

ETH\_CHECKSUM\_BY\_HARDWARE

ETH\_CHECKSUM\_BY\_SOFTWARE

***ETH Checksum Offload***

ETH\_CHECKSUMOFFLOAD\_ENABLE

ETH\_CHECKSUMOFFLOAD\_DISABLE

***ETH Deferral Check***

ETH\_DEFFERRALCHECK\_ENABLE

ETH\_DEFFERRALCHECK\_DISABLE

***ETH Destination Addr Filter***

ETH\_DESTINATIONADDRFILTER\_NORMAL

ETH\_DESTINATIONADDRFILTER\_INVERSE

***ETH DMA Arbitration***

ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_1\_1



ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_2\_1

ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_3\_1

ETH\_DMAARBITRATION\_ROUNDROBIN\_RXTX\_4\_1

ETH\_DMAARBITRATION\_RXPRIORTX

***ETH DMA 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_QUEUING	Running - queuing the receive frame into host memory

**ETH DMA RX Descriptor**

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

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

	d Checksum Error: Rx MAC address matched/ Rx Payload Checksum Error
ETH_DMARXDESC_DIC	Disable Interrupt on Completion
ETH_DMARXDESC_RBS2	Receive Buffer2 Size
ETH_DMARXDESC_RER	Receive End of Ring
ETH_DMARXDESC_RCH	Second Address Chained
ETH_DMARXDESC_RBS1	Receive Buffer1 Size
ETH_DMARXDESC_B1AP	Buffer1 Address Pointer
ETH_DMARXDESC_B2AP	Buffer2 Address Pointer
ETH_DMAPTPRXDESC_PTPV	
ETH_DMAPTPRXDESC_PTPFT	
ETH_DMAPTPRXDESC_PTPMT	
ETH_DMAPTPRXDESC_PTPMT_SYNC	
ETH_DMAPTPRXDESC_PTPMT_FOLLOWUP	
ETH_DMAPTPRXDESC_PTPMT_DELAYREQ	
ETH_DMAPTPRXDESC_PTPMT_DELAYRESP	
ETH_DMAPTPRXDESC_PTPMT_PDELAYREQ_ANNOUNCE	
ETH_DMAPTPRXDESC_PTPMT_PDELAYRESP_MANAG	
ETH_DMAPTPRXDESC_PTPMT_PDELAYRESPFOLLOWUP_SIGNA L	
ETH_DMAPTPRXDESC_IPV6PR	
ETH_DMAPTPRXDESC_IPV4PR	
ETH_DMAPTPRXDESC_IPCB	
ETH_DMAPTPRXDESC_IPPE	
ETH_DMAPTPRXDESC_IPHE	
ETH_DMAPTPRXDESC_IPPT	
ETH_DMAPTPRXDESC_IPPT_UDP	
ETH_DMAPTPRXDESC_IPPT_TCP	
ETH_DMAPTPRXDESC_IPPT_ICMP	
ETH_DMAPTPRXDESC_RTSL	

ETH\_DMAPTPRXDESC\_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**

ETH\_DMA\_TRANSMITPROCESS\_STOPPED Stopped - Reset or Stop Tx Command issued

ETH\_DMA\_TRANSMITPROCESS\_FETCHING Running - fetching the Tx descriptor

ETH\_DMA\_TRANSMITPROCESS\_WAITING Running - waiting for status

ETH\_DMA\_TRANSMITPROCESS\_READING Running - reading the data from host memory

ETH\_DMA\_TRANSMITPROCESS\_SUSPENDED Suspended - Tx Descriptor unavailable

ETH\_DMA\_TRANSMITPROCESS\_CLOSING Running - closing Rx descriptor

**ETH DMA TX Descriptor**

ETH\_DMATXDESC\_OWN OWN bit: descriptor is owned by DMA engine

ETH\_DMATXDESC\_IC Interrupt on Completion

ETH\_DMATXDESC\_LS Last Segment

ETH\_DMATXDESC\_FS First Segment

ETH\_DMATXDESC\_DC Disable CRC

ETH\_DMATXDESC\_DP Disable Padding

ETH\_DMATXDESC\_TTSE Transmit Time Stamp Enable

ETH\_DMATXDESC\_CIC Checksum Insertion Control: 4 cases

ETH\_DMATXDESC\_CIC\_BYPASS Do Nothing: Checksum Engine is bypassed

ETH\_DMATXDESC\_CIC\_IPV4HEADER IPV4 header Checksum Insertion

ETH\_DMATXDESC\_CIC\_TCPUDPICMP\_SEGMENT TCP/UDP/ICMP Checksum Insertion calculated over segment only

ETH\_DMATXDESC\_CIC\_TCPUDPICMP\_FULL TCP/UDP/ICMP Checksum Insertion fully calculated

ETH\_DMATXDESC\_TER Transmit End of Ring

ETH\_DMATXDESC\_TCH Second Address Chained

ETH\_DMATXDESC\_TTSS Tx Time Stamp Status

ETH\_DMATXDESC\_IHE IP Header Error

ETH\_DMATXDESC\_ES Error summary: OR of the following bits: UE || ED || EC || LCO || NC || LCA || FF || JT

ETH_DMATXDESC_JT	Jabber Timeout
ETH_DMATXDESC_FF	Frame Flushed: DMA/MTL flushed the frame due to SW flush
ETH_DMATXDESC_PCE	Payload Checksum Error
ETH_DMATXDESC_LCA	Loss of Carrier: carrier lost during transmission
ETH_DMATXDESC_NC	No Carrier: no carrier signal from the transceiver
ETH_DMATXDESC_LCO	Late Collision: transmission aborted due to collision
ETH_DMATXDESC_EC	Excessive Collision: transmission aborted after 16 collisions
ETH_DMATXDESC_VF	VLAN Frame
ETH_DMATXDESC_CC	Collision Count
ETH_DMATXDESC_ED	Excessive Deferral
ETH_DMATXDESC_UF	Underflow Error: late data arrival from the memory
ETH_DMATXDESC_DB	Deferred Bit
ETH_DMATXDESC_TBS2	Transmit Buffer2 Size
ETH_DMATXDESC_TBS1	Transmit Buffer1 Size
ETH_DMATXDESC_B1AP	Buffer1 Address Pointer
ETH_DMATXDESC_B2AP	Buffer2 Address Pointer
ETH_DMAPTPTXDESC_TTSL	
ETH_DMAPTPTXDESC_TTSH	
<b>ETH DMA Tx descriptor Checksum Insertion Control</b>	
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
<b>ETH DMA Tx descriptor segment</b>	
ETH_DMATXDESC_LASTSEGMENTS	Last Segment
ETH_DMATXDESC_FIRSTSEGMENT	First Segment
<b>ETH Drop TCP IP Checksum Error Frame</b>	
ETH_DROPTCPIPCHECKSUMERRORFRAME_ENABLE	

ETH\_DROPTCPIPCHECKSUMERRORFRAME\_DISABLE

**ETH Duplex Mode**

ETH\_MODE\_FULLLDUPLEX

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:**

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

- None

**Description:**

- Disables the specified DMA Rx Desc receive interrupt.

**Parameters:**

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

**Return value:**

- None

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

**Description:**

- Set the specified DMA Rx Desc Own bit.

**Parameters:**

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

**Return value:**

- None

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

**Description:**

- Returns the specified ETHERNET DMA Tx Desc collision count.

**Parameters:**

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

**Return value:**

- The: Transmit descriptor collision counter value.

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

**Description:**

- Set the specified DMA Tx Desc Own bit.

**Parameters:**

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

**Return value:**

- None

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

**Description:**

- Enables the specified DMA Tx Desc Transmit interrupt.

**Parameters:**

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

**Return value:**

- None

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

**Description:**

- Disables the specified DMA Tx Desc



Transmit interrupt.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_DMATXDESC_CHECKSUM_INSERTION`

**Description:**

- Selects the specified ETHERNET DMA Tx Desc Checksum Insertion.

**Parameters:**

- `__HANDLE__`: ETH Handle
- `__CHECKSUM__`: specifies is the DMA Tx desc checksum insertion. This parameter can be one of the following values:
  - `ETH_DMATXDESC_CHECKSUMBYPASS`: Checksum bypass
  - `ETH_DMATXDESC_CHECKSUMIPv4HEADER`: IPv4 header checksum
  - `ETH_DMATXDESC_CHECKSUMTCPUDPICMPSEGMENT`: TCP/UDP/ICMP checksum. Pseudo header checksum is assumed to be present
  - `ETH_DMATXDESC_CHECKSUMTCPUDPICMPFULL`: TCP/UDP/ICMP checksum fully in hardware including pseudo header

**Return value:**

- None

`__HAL_ETH_DMATXDESC_CRC_ENABLE`

**Description:**

- Enables the DMA Tx Desc CRC.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_DMATXDESC_CRC_DISABLE`

**Description:**

- Disables the DMA Tx Desc CRC.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_DMATXDESC_SHORT_FRAME_PADDING_ENABLE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_DMATXDESC_SHORT_FRAME_PADDING_DISABLE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_MAC_ENABLE_IT`

**Description:**

- Enables the specified ETHERNET MAC interrupts.

**Parameters:**

- `__HANDLE__`: : ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET MAC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `ETH_MAC_IT_TST` : Time stamp trigger interrupt
  - `ETH_MAC_IT_PMT` : PMT interrupt

**Return value:**

- None

`__HAL_ETH_MAC_DISABLE_IT`

**Description:**

- Disables the specified ETHERNET MAC interrupts.

**Parameters:**

- `__HANDLE__`: : ETH Handle
- `__INTERRUPT__`: specifies the ETHERNET MAC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - `ETH_MAC_IT_TST` : Time stamp trigger interrupt
  - `ETH_MAC_IT_PMT` : PMT interrupt

**Return value:**

`__HAL_ETH_INITIATE_PAUSE_CONTROL_FRAME`

- None

**Description:**

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

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_GET_FLOW_CONTROL_BUSY_STATUS`

**Description:**

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

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

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

`__HAL_ETH_BACK_PRESSURE_ACTIVATION_ENABLE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_BACK_PRESSURE_ACTIVATION_DISABLE`

**Description:**

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

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_MAC_GET_FLAG`

**Description:**

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

**Parameters:**

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

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

**Return value:**

- The: state of ETHERNET MAC flag.

**Description:**

- Enables the specified ETHERNET DMA interrupts.

**Parameters:**

- \_\_HANDLE\_\_: : ETH Handle
- \_\_INTERRUPT\_\_: specifies the ETHERNET DMA interrupt sources to be enabled

**Return value:**

- None

**Description:**

- Disables the specified ETHERNET DMA interrupts.

**Parameters:**

- \_\_HANDLE\_\_: : ETH Handle
- \_\_INTERRUPT\_\_: specifies the ETHERNET DMA interrupt sources to be disabled.

**Return value:**

- None

**Description:**

- Clears the ETHERNET DMA IT pending bit.

**Parameters:**

- \_\_HANDLE\_\_: : ETH Handle
- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear.

**Return value:**

- None

**Description:**

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

**Parameters:**

\_\_HAL\_ETH\_DMA\_ENABLE\_IT

\_\_HAL\_ETH\_DMA\_DISABLE\_IT

\_\_HAL\_ETH\_DMA\_CLEAR\_IT

\_\_HAL\_ETH\_DMA\_GET\_FLAG

`__HAL_ETH_DMA_CLEAR_FLAG`

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

**Return value:**

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

**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:**

---

`UP_ENABLE`

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

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_GLOBAL_UNICAST_WAKEUP_DISABLE`**Description:**

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

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_WAKEUP_FRAME_DETECTION_ENABLE`**Description:**

- Enables the MAC Wake-Up Frame Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_WAKEUP_FRAME_DETECTION_DISABLE`**Description:**

- Disables the MAC Wake-Up Frame Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_MAGIC_PACKET_DETECTION_ENABLE`**Description:**

- Enables the MAC Magic Packet Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_MAGIC_PACKET_DETECTION_DISABLE`**Description:**

- Disables the MAC Magic Packet

Detection.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_POWER_DOWN_ENABLE`

**Description:**

- Enables the MAC Power Down.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_POWER_DOWN_DISABLE`

**Description:**

- Disables the MAC Power Down.

**Parameters:**

- `__HANDLE__`: ETH Handle

**Return value:**

- None

`__HAL_ETH_GET_PMT_FLAG_STATUS`

**Description:**

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

**Parameters:**

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

**Return value:**

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

`__HAL_ETH_MMC_COUNTER_FULL_PR  
ESET`

**Description:**

- Preset and Initialize the MMC counters to almost-full value: `0xFFFF_FFF0` (full - 16)

**Parameters:**

`__HAL_ETH_MMC_COUNTER_HALF_PR  
ESET`

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

**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

`__HAL_ETH_ETH_MMC_COUNTER_ROLLOVER_DISABLE`

**Description:**

- Disables the MMC Counter Stop Rollover.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_MMC_COUNTERS_RESET`

**Description:**

- Resets the MMC Counters.

**Parameters:**

- `__HANDLE__`: ETH Handle.

**Return value:**

- None

`__HAL_ETH_MMC_RX_IT_ENABLE`

**Description:**

- Enables the specified ETHERNET MMC Rx interrupts.

**Parameters:**

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

**Return value:**

- None

`__HAL_ETH_MMC_RX_IT_DISABLE`

**Description:**

- Disables the specified ETHERNET MMC Rx interrupts.

**Parameters:**

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

**Return value:**

- None

**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_TGFMS`: 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

**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

`__HAL_ETH_MMC_TX_IT_ENABLE``__HAL_ETH_MMC_TX_IT_DISABLE`

- 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_I`  
`T`

**Description:**

- Enables the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_I`  
`T`

**Description:**

- Disables the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_E`  
`VENT`

**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_FL`  
`AG`

**Description:**

- Clear flag of the ETH External interrupt line.

**Return value:**

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_R`  
`ISING_EDGE_TRIGGER`

**Description:**

- Enables rising edge trigger to the ETH

External interrupt line.

**Return value:**

- None

**Description:**

- Disables the rising edge trigger to the ETH External interrupt line.

**Return value:**

- None

**Description:**

- Enables falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

**Description:**

- Disables falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

**Description:**

- Enables rising/falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

**Description:**

- Disables rising/falling edge trigger to the ETH External interrupt line.

**Return value:**

- None

**Description:**

- Generate a Software interrupt on selected EXTI line.

**Return value:**

- None.

`__HAL_ETH_WAKEUP_EXTI_DISABLE_RISING_EDGE_TRIGGER`

`__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLING_EDGE_TRIGGER`

`__HAL_ETH_WAKEUP_EXTI_DISABLE_FALLING_EDGE_TRIGGER`

`__HAL_ETH_WAKEUP_EXTI_ENABLE_FALLINGRISING_TRIGGER`

`__HAL_ETH_WAKEUP_EXTI_DISABLE_FALLINGRISING_TRIGGER`

`__HAL_ETH_WAKEUP_EXTI_GENERATE_SWIT`

**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\_GENERATING\_PCF  
ETH\_MAC\_TRANSMITFRAMECONTROLLER\_TRANSFERRING  
ETH\_MAC\_MII\_TRANSMIT\_ACTIVE  
ETH\_MAC\_RXFIFO\_EMPTY  
ETH\_MAC\_RXFIFO\_BELOW\_THRESHOLD  
ETH\_MAC\_RXFIFO\_ABOVE\_THRESHOLD  
ETH\_MAC\_RXFIFO\_FULL  
ETH\_MAC\_READCONTROLLER\_IDLE  
ETH\_MAC\_READCONTROLLER\_READING\_DATA  
ETH\_MAC\_READCONTROLLER\_READING\_STATUS  
ETH\_MAC\_READCONTROLLER\_  
ETH\_MAC\_RXFIFO\_WRITE\_ACTIVE  
ETH\_MAC\_SMALL\_FIFO\_NOTACTIVE  
ETH\_MAC\_SMALL\_FIFO\_READ\_ACTIVE  
ETH\_MAC\_SMALL\_FIFO\_WRITE\_ACTIVE  
ETH\_MAC\_SMALL\_FIFO\_RW\_ACTIVE  
ETH\_MAC\_MII\_RECEIVE\_PROTOCOL\_ACTIVE

**ETH MAC Flags**

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

**ETH MAC Interrupts**

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

**ETH Media Interface**

ETH_MEDIA_INTERFACE_MII
ETH_MEDIA_INTERFACE_RMII

**ETH MMC Rx Interrupts**

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

**ETH MMC Tx Interrupts**

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

**ETH Multicast Frames Filter**

ETH_MULTICASTFRAMESFILTER_PERFECTHASHTABLE
ETH_MULTICASTFRAMESFILTER_HASHTABLE
ETH_MULTICASTFRAMESFILTER_PERFECT
ETH_MULTICASTFRAMESFILTER_NONE

**ETH Pass Control Frames**

ETH_PASSCONTROLFRAMES_BLOCKALL	MAC filters all control frames from reaching the application
ETH_PASSCONTROLFRAMES_FORWARDALL	MAC forwards all control frames to application even if they fail the Address Filter

ETH\_PASSCONTROLFRAMES\_FORWARDPASSEDADDRFILTER    MAC forwards control frames that pass the Address Filter.

**ETH Pause Low Threshold**

ETH\_PAUSELOWTHRESHOLD\_MINUS4    Pause time minus 4 slot times  
ETH\_PAUSELOWTHRESHOLD\_MINUS28    Pause time minus 28 slot times  
ETH\_PAUSELOWTHRESHOLD\_MINUS144    Pause time minus 144 slot times  
ETH\_PAUSELOWTHRESHOLD\_MINUS256    Pause time minus 256 slot times

**ETH PMT Flags**

ETH\_PMT\_FLAG\_WUFFRPR    Wake-Up Frame Filter Register Pointer Reset  
ETH\_PMT\_FLAG\_WUFR    Wake-Up Frame Received  
ETH\_PMT\_FLAG\_MPR    Magic Packet Received

**ETH Private Constants**

LINKED\_STATE\_TIMEOUT\_VALUE  
AUTONEGO\_COMPLETED\_TIMEOUT\_VALUE

**ETH Private Defines**

ETH\_REG\_WRITE\_DELAY  
ETH\_SUCCESS  
ETH\_ERROR  
ETH\_DMATXDESC\_COLLISION\_COUNTSHIFT  
ETH\_DMATXDESC\_BUFFER2\_SIZESHIFT  
ETH\_DMARXDESC\_FRAME\_LENGTHSHIFT  
ETH\_DMARXDESC\_BUFFER2\_SIZESHIFT  
ETH\_DMARXDESC\_FRAMELENGTHSHIFT  
ETH\_MAC\_ADDR\_HBASE  
ETH\_MAC\_ADDR\_LBASE  
ETH\_MACMIAR\_CR\_MASK  
ETH\_MACCR\_CLEAR\_MASK  
ETH\_MACFCR\_CLEAR\_MASK  
ETH\_DMAOMR\_CLEAR\_MASK  
ETH\_WAKEUP\_REGISTER\_LENGTH  
ETH\_DMA\_RX\_OVERFLOW\_MISSEDFRAMES\_COUNTERSHIFT

**ETH Private Macros**

IS\_ETH\_PHY\_ADDRESS  
IS\_ETH\_AUTONEGOTIATION  
IS\_ETH\_SPEED



IS\_ETH\_DUPLEX\_MODE  
IS\_ETH\_DUPLEX\_MODE  
IS\_ETH\_RX\_MODE  
IS\_ETH\_RX\_MODE  
IS\_ETH\_RX\_MODE  
IS\_ETH\_CHECKSUM\_MODE  
IS\_ETH\_MEDIA\_INTERFACE  
IS\_ETH\_WATCHDOG  
IS\_ETH\_JABBER  
IS\_ETH\_INTER\_FRAME\_GAP  
IS\_ETH\_CARRIER\_SENSE  
IS\_ETH\_RECEIVE\_OWN  
IS\_ETH\_LOOPBACK\_MODE  
IS\_ETH\_CHECKSUM\_OFFLOAD  
IS\_ETH\_RETRY\_TRANSMISSION  
IS\_ETH\_AUTOMATIC\_PADCRC\_STRIP  
IS\_ETH\_BACKOFF\_LIMIT  
IS\_ETH\_DEFERRAL\_CHECK  
IS\_ETH\_RECEIVE\_ALL  
IS\_ETH\_SOURCE\_ADDR\_FILTER  
IS\_ETH\_CONTROL\_FRAMES  
IS\_ETH\_BROADCAST\_FRAMES\_RECEPTION  
IS\_ETH\_DESTINATION\_ADDR\_FILTER  
IS\_ETH\_PROMISCUOUS\_MODE  
IS\_ETH\_MULTICAST\_FRAMES\_FILTER  
IS\_ETH\_UNICAST\_FRAMES\_FILTER  
IS\_ETH\_PAUSE\_TIME  
IS\_ETH\_ZEROQUANTA\_PAUSE  
IS\_ETH\_PAUSE\_LOW\_THRESHOLD  
IS\_ETH\_UNICAST\_PAUSE\_FRAME\_DETECT  
IS\_ETH\_RECEIVE\_FLOWCONTROL  
IS\_ETH\_TRANSMIT\_FLOWCONTROL  
IS\_ETH\_VLAN\_TAG\_COMPARISON  
IS\_ETH\_VLAN\_TAG\_IDENTIFIER  
IS\_ETH\_MAC\_ADDRESS0123  
IS\_ETH\_MAC\_ADDRESS123

IS\_ETH\_MAC\_ADDRESS\_FILTER  
IS\_ETH\_MAC\_ADDRESS\_MASK  
IS\_ETH\_DROP\_TCPIP\_CHECKSUM\_FRAME  
IS\_ETH\_RECEIVE\_STORE\_FORWARD  
IS\_ETH\_FLUSH\_RECEIVE\_FRAME  
IS\_ETH\_TRANSMIT\_STORE\_FORWARD  
IS\_ETH\_TRANSMIT\_THRESHOLD\_CONTROL  
IS\_ETH\_FORWARD\_ERROR\_FRAMES  
IS\_ETH\_FORWARD\_UNDERSIZED\_GOOD\_FRAMES  
IS\_ETH\_RECEIVE\_THRESHOLD\_CONTROL  
IS\_ETH\_SECOND\_FRAME\_OPERATE  
IS\_ETH\_ADDRESS\_ALIGNED\_BEATS  
IS\_ETH\_FIXED\_BURST  
IS\_ETH\_RXDMA\_BURST\_LENGTH  
IS\_ETH\_TXDMA\_BURST\_LENGTH  
IS\_ETH\_DMA\_DESC\_SKIP\_LENGTH  
IS\_ETH\_DMA\_ARBITRATION\_ROUNDROBIN\_RXTX  
IS\_ETH\_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

***ETH Receive Flow Control***

ETH\_RECEIVEFLOWCONTROL\_ENABLE

ETH\_RECEIVEFLOWCONTROL\_DISABLE

***ETH Receive Own***

ETH\_RECEIVEOWN\_ENABLE

ETH\_RECEIVEOWN\_DISABLE

***ETH Receive Store Forward***

ETH\_RECEIVESTOREFORWARD\_ENABLE

ETH\_RECEIVESTOREFORWARD\_DISABLE

***ETH Receive Threshold Control***

ETH\_RECEIVEDTHRESHOLDCONTROL\_64BYTES threshold level of the MTL  
Receive FIFO is 64 Bytes

ETH\_RECEIVEDTHRESHOLDCONTROL\_32BYTES threshold level of the MTL  
Receive FIFO is 32 Bytes

ETH\_RECEIVEDTHRESHOLDCONTROL\_96BYTES threshold level of the MTL  
Receive FIFO is 96 Bytes

ETH\_RECEIVEDTHRESHOLDCONTROL\_128BYTES threshold level of the MTL  
Receive FIFO is 128 Bytes

***ETH Retry Transmission***

ETH\_RETRYTRANSMISSION\_ENABLE

ETH\_RETRYTRANSMISSION\_DISABLE

***ETH Rx DMA Burst Length***

ETH\_RXDMABURSTLENGTH\_1BEAT maximum number of beats to be  
transferred in one RxDMA transaction  
is 1

ETH\_RXDMABURSTLENGTH\_2BEAT maximum number of beats to be  
transferred in one RxDMA transaction  
is 2

ETH\_RXDMABURSTLENGTH\_4BEAT maximum number of beats to be  
transferred in one RxDMA transaction  
is 4

ETH\_RXDMABURSTLENGTH\_8BEAT maximum number of beats to be  
transferred in one RxDMA transaction  
is 8

ETH\_RXDMABURSTLENGTH\_16BEAT maximum number of beats to be  
transferred in one RxDMA transaction

	is 16
ETH_RXDMABURSTLENGTH_32BEAT	maximum number of beats to be transferred in one RxDMA transaction is 32
ETH_RXDMABURSTLENGTH_4XPBL_4BEAT	maximum number of beats to be transferred in one RxDMA transaction is 4
ETH_RXDMABURSTLENGTH_4XPBL_8BEAT	maximum number of beats to be transferred in one RxDMA transaction is 8
ETH_RXDMABURSTLENGTH_4XPBL_16BEAT	maximum number of beats to be transferred in one RxDMA transaction is 16
ETH_RXDMABURSTLENGTH_4XPBL_32BEAT	maximum number of beats to be transferred in one RxDMA transaction is 32
ETH_RXDMABURSTLENGTH_4XPBL_64BEAT	maximum number of beats to be transferred in one RxDMA transaction is 64
ETH_RXDMABURSTLENGTH_4XPBL_128BEAT	maximum number of beats to be transferred in one RxDMA transaction is 128

**ETH Rx Mode**

ETH\_RXPOLLING\_MODE

ETH\_RXINTERRUPT\_MODE

**ETH Second Frame Operate**

ETH\_SECONDFRAMEOPERARTE\_ENABLE

ETH\_SECONDFRAMEOPERARTE\_DISABLE

**ETH Source Addr Filter**

ETH\_SOURCEADDRFILTER\_NORMAL\_ENABLE

ETH\_SOURCEADDRFILTER\_INVERSE\_ENABLE

ETH\_SOURCEADDRFILTER\_DISABLE

**ETH Speed**

ETH\_SPEED\_10M

ETH\_SPEED\_100M

**ETH Transmit Flow Control**

ETH\_TRANSMITFLOWCONTROL\_ENABLE

ETH\_TRANSMITFLOWCONTROL\_DISABLE

**ETH Transmit Store Forward**

ETH\_TRANSMITSTOREFORWARD\_ENABLE

ETH\_TRANSMITSTOREFORWARD\_DISABLE

**ETH Transmit Threshold Control**

ETH_TRANSMITTHRESHOLDCONTROL_64BYTES	threshold level of the MTL Transmit FIFO is 64 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_128BYTES	threshold level of the MTL Transmit FIFO is 128 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_192BYTES	threshold level of the MTL Transmit FIFO is 192 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_256BYTES	threshold level of the MTL Transmit FIFO is 256 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_40BYTES	threshold level of the MTL Transmit FIFO is 40 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_32BYTES	threshold level of the MTL Transmit FIFO is 32 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_24BYTES	threshold level of the MTL Transmit FIFO is 24 Bytes
ETH_TRANSMITTHRESHOLDCONTROL_16BYTES	threshold level of the MTL Transmit FIFO is 16 Bytes

**ETH Tx DMA Burst Length**

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

---

ETH_TXDMABURSTLENGTH_4XPBL_32BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 32
ETH_TXDMABURSTLENGTH_4XPBL_64BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 64
ETH_TXDMABURSTLENGTH_4XPBL_128BEAT	maximum number of beats to be transferred in one TxDMA (or both) transaction is 128

***ETH Unicast Frames Filter***

ETH\_UNICASTFRAMESFILTER\_PERFECTHASHTABLE

ETH\_UNICASTFRAMESFILTER\_HASHTABLE

ETH\_UNICASTFRAMESFILTER\_PERFECT

***ETH Unicast Pause Frame Detect***

ETH\_UNICASTPAUSEFRAMEDETECT\_ENABLE

ETH\_UNICASTPAUSEFRAMEDETECT\_DISABLE

***ETH VLAN Tag Comparison***

ETH\_VLANTAGCOMPARISON\_12BIT

ETH\_VLANTAGCOMPARISON\_16BIT

***ETH Watchdog***

ETH\_WATCHDOG\_ENABLE

ETH\_WATCHDOG\_DISABLE

***ETH Zero Quanta Pause***

ETH\_ZEROQUANTAPAUSE\_ENABLE

ETH\_ZEROQUANTAPAUSE\_DISABLE

## 21 HAL FLASH Generic Driver

### 21.1 FLASH Firmware driver registers structures

#### 21.1.1 FLASH\_ProcessTypeDef

##### Data Fields

- *\_\_IO FLASH\_ProcedureTypeDef ProcedureOnGoing*
- *\_\_IO uint32\_t NbSectorsToErase*
- *\_\_IO uint8\_t VoltageForErase*
- *\_\_IO uint32\_t Sector*
- *\_\_IO uint32\_t 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::Address*
- *HAL\_LockTypeDef FLASH\_ProcessTypeDef::Lock*
- *\_\_IO uint32\_t FLASH\_ProcessTypeDef::ErrorCode*

### 21.2 FLASH Firmware driver API description

#### 21.2.1 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

#### 21.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32F7xx 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



For any Flash memory program operation (erase or program), the CPU clock frequency (HCLK) must be at least 1MHz.



The contents of the Flash memory are not guaranteed if a device reset occurs during a Flash memory operation.



Any attempt to read the Flash memory while it is being written or erased, causes the bus to stall. Read operations are processed correctly once the program operation has completed. This means that code or data fetches cannot be performed while a write/erase operation is ongoing.

### 21.2.3 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\(\)\*](#)

### 21.2.4 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\(\)](#)

### 21.2.5 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\(\)](#)

### 21.2.6 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	<ul style="list-style-type: none"> <li>• HAL_StatusTypeDef HAL Status</li> </ul>

### 21.2.7 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	<ul style="list-style-type: none"> <li>• HAL Status</li> </ul>

### 21.2.8 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>

### 21.2.9 HAL\_FLASH\_EndOfOperationCallback

Function Name	<b>void HAL_FLASH_EndOfOperationCallback (uint32_t</b>
---------------	--

**ReturnValue)**

Function Description FLASH end of operation interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure  
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  
Mass Erase : No return value expected

Return values

- None

**21.2.10 HAL\_FLASH\_OperationErrorCallback**

Function Name **void HAL\_FLASH\_OperationErrorCallback (uint32\_t ReturnValue)**

Function Description FLASH operation error interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure  
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  
Mass Erase : No return value expected

Return values

- None

**21.2.11 HAL\_FLASH\_Unlock**

Function Name **HAL\_StatusTypeDef HAL\_FLASH\_Unlock (void )**

Function Description Unlock the FLASH control register access.

Return values

- HAL Status

**21.2.12 HAL\_FLASH\_Lock**

Function Name **HAL\_StatusTypeDef HAL\_FLASH\_Lock (void )**

Function Description Locks the FLASH control register access.

Return values

- HAL Status

**21.2.13 HAL\_FLASH\_OB\_Unlock**

Function Name **HAL\_StatusTypeDef HAL\_FLASH\_OB\_Unlock (void )**

Function Description Unlock the FLASH Option Control Registers access.

Return values

- HAL Status

**21.2.14 HAL\_FLASH\_OB\_Lock**

Function Name **HAL\_StatusTypeDef HAL\_FLASH\_OB\_Lock (void )**

Function Description Lock the FLASH Option Control Registers access.

Return values

- HAL Status

**21.2.15 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>

### 21.2.16 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: FLASH_ERROR_ERS: FLASH Erasing Sequence error flag FLASH_ERROR_PGP: FLASH Programming Parallelism error flag FLASH_ERROR_PGA: FLASH Programming Alignment error flag FLASH_ERROR_WRP: FLASH Write protected error flag FLASH_ERROR_OPERATION: FLASH operation Error flag</li> </ul>

### 21.2.17 FLASH\_WaitForLastOperation

Function Name	<b>HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t 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>

## 21.3 FLASH Firmware driver defines

### 21.3.1 FLASH

#### **FLASH Error Code**

HAL_FLASH_ERROR_NONE	No error
HAL_FLASH_ERROR_ERS	Programming Sequence error
HAL_FLASH_ERROR_PGP	Programming Parallelism error
HAL_FLASH_ERROR_PGA	Programming Alignment error
HAL_FLASH_ERROR_WRP	Write protection error
HAL_FLASH_ERROR_OPERATION	Operation Error

#### **FLASH Exported Macros**

**\_\_HAL\_FLASH\_SET\_LATENCY**

#### **Description:**

- Set the FLASH Latency.

#### **Parameters:**

- **\_\_LATENCY\_\_**: FLASH Latency  
The value of this parameter depend on device used within the same series

#### **Return value:**

\_\_HAL\_FLASH\_PREFETCH\_BUFFER\_ENABLE

- none

**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\_ART\_ENABLE

**Description:**

- Enable the FLASH Adaptive Real-Time memory accelerator.

**Return value:**

- none

**Notes:**

- The ART accelerator is available only for flash access on ITCM interface.

\_\_HAL\_FLASH\_ART\_DISABLE

**Description:**

- Disable the FLASH Adaptive Real-Time memory accelerator.

**Return value:**

- none

\_\_HAL\_FLASH\_ART\_RESET

**Description:**

- Resets the FLASH Adaptive Real-Time memory accelerator.

**Return value:**

- None

**Notes:**

- This function must be used only when the Adaptive Real-Time memory accelerator 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

**Description:**

- Disable the specified FLASH interrupt.

**Parameters:**

- `__INTERERRUPT__`: 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

**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\_ERSERR : FLASH Erasing Sequence error flag
  - FLASH\_FLAG\_BSY : FLASH Busy flag

**Return value:**

- The: new state of `__FLAG__` (SET

`__HAL_FLASH_DISABLE_IT``__HAL_FLASH_GET_FLAG`

`__HAL_FLASH_CLEAR_FLAG`

or RESET).

**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_ERSERR` : FLASH Erasing Sequence error flag

**Return value:**

- none

**FLASH Flag definition**

<code>FLASH_FLAG_EOP</code>	FLASH End of Operation flag
<code>FLASH_FLAG_OPERR</code>	FLASH operation Error flag
<code>FLASH_FLAG_WRPERR</code>	FLASH Write protected error flag
<code>FLASH_FLAG_PGAERR</code>	FLASH Programming Alignment error flag
<code>FLASH_FLAG_PGPERR</code>	FLASH Programming Parallelism error flag
<code>FLASH_FLAG_ERSERR</code>	FLASH Erasing Sequence error flag
<code>FLASH_FLAG_BSY</code>	FLASH Busy flag

**FLASH Interrupt definition**

<code>FLASH_IT_EOP</code>	End of FLASH Operation Interrupt source
<code>FLASH_IT_ERR</code>	Error Interrupt source

**FLASH Private macros to check input parameters**`IS_FLASH_TYPEPROGRAM`**FLASH Keys**

`FLASH_KEY1`  
`FLASH_KEY2`  
`FLASH_OPT_KEY1`

FLASH\_OPT\_KEY2

**FLASH Latency**

FLASH_LATENCY_0	FLASH Zero Latency cycle
FLASH_LATENCY_1	FLASH One Latency cycle
FLASH_LATENCY_2	FLASH Two Latency cycles
FLASH_LATENCY_3	FLASH Three Latency cycles
FLASH_LATENCY_4	FLASH Four Latency cycles
FLASH_LATENCY_5	FLASH Five Latency cycles
FLASH_LATENCY_6	FLASH Six Latency cycles
FLASH_LATENCY_7	FLASH Seven Latency cycles
FLASH_LATENCY_8	FLASH Eight Latency cycles
FLASH_LATENCY_9	FLASH Nine Latency cycles
FLASH_LATENCY_10	FLASH Ten Latency cycles
FLASH_LATENCY_11	FLASH Eleven Latency cycles
FLASH_LATENCY_12	FLASH Twelve Latency cycles
FLASH_LATENCY_13	FLASH Thirteen Latency cycles
FLASH_LATENCY_14	FLASH Fourteen Latency cycles
FLASH_LATENCY_15	FLASH Fifteen Latency cycles

**FLASH Private Constants**

SECTOR\_MASK

FLASH\_TIMEOUT\_VALUE

**FLASH Program Parallelism**

FLASH\_PSIZE\_BYTE

FLASH\_PSIZE\_HALF\_WORD

FLASH\_PSIZE\_WORD

FLASH\_PSIZE\_DOUBLE\_WORD

CR\_PSIZE\_MASK

**FLASH Type Program**

FLASH_TYPEPROGRAM_BYTE	Program byte (8-bit) at a specified address
FLASH_TYPEPROGRAM_HALFWORD	Program a half-word (16-bit) at a specified address
FLASH_TYPEPROGRAM_WORD	Program a word (32-bit) at a specified address
FLASH_TYPEPROGRAM_DOUBLEWORD	Program a double word (64-bit) at a specified address

## 22 HAL FLASH Extension Driver

### 22.1 FLASHEx Firmware driver registers structures

#### 22.1.1 FLASH\_EraseInitTypeDef

##### Data Fields

- *uint32\_t TypeErase*
- *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::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](#)

#### 22.1.2 FLASH\_OBProgramInitTypeDef

##### Data Fields

- *uint32\_t OptionType*
- *uint32\_t WRPState*
- *uint32\_t WRPSector*
- *uint32\_t RDPLLevel*
- *uint32\_t BORLevel*
- *uint32\_t USERConfig*
- *uint32\_t BootAddr0*
- *uint32\_t BootAddr1*

##### 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::RDPLLevel`**  
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\*](#)
- **`uint32_t FLASH_OBProgramInitTypeDef::USERConfig`**  
Program the FLASH User Option Byte: WWDG\_SW / IWDG\_SW / RST\_STOP / RST\_STDBY / IWDG\_FREEZE\_STOP / IWDG\_FREEZE\_SANDBY.
- **`uint32_t FLASH_OBProgramInitTypeDef::BootAddr0`**  
Boot base address when Boot pin = 0. This parameter can be a value of [\*FLASHEx\\_Boot\\_Address\*](#)
- **`uint32_t FLASH_OBProgramInitTypeDef::BootAddr1`**  
Boot base address when Boot pin = 1. This parameter can be a value of [\*FLASHEx\\_Boot\\_Address\*](#)

## 22.2 FLASHEx Firmware driver API description

### 22.2.1 Flash Extension features

Comparing to other previous devices, the FLASH interface for STM32F727xx/437xx and devices contains the following additional features

- Capacity up to 2 Mbyte with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank memory organization
- PCROP protection for all banks

### 22.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32F7xx devices. It includes

1. FLASH Memory Erase functions:
  - Lock and Unlock the FLASH interface using `HAL_FLASH_Unlock()` and `HAL_FLASH_Lock()` functions
  - Erase function: Erase sector, erase all sectors
  - There are two modes of erase :
    - Polling Mode using `HAL_FLASHEx_Erase()`
    - Interrupt Mode using `HAL_FLASHEx_Erase_IT()`
2. Option Bytes Programming functions: Use `HAL_FLASHEx_OBProgram()` to :
  - Set/Reset the write protection
  - Set the Read protection Level
  - Set the BOR level
  - Program the user Option Bytes
3. Advanced Option Bytes Programming functions: Use `HAL_FLASHEx_AdvOBProgram()` to :
  - Extended space (bank 2) erase function
  - Full FLASH space (2 Mo) erase (bank 1 and bank 2)

- Dual Boot activation
- Write protection configuration for bank 2
- PCROP protection configuration and control for both banks

### 22.2.3 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\(\)\*](#)

### 22.2.4 HAL\_FLASHEx\_Erase

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase</b> (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * SectorError)
Function Description	Perform a mass erase or erase the specified FLASH memory sectors.
Parameters	<ul style="list-style-type: none"> <li>• <b>pEraseInit:</b> pointer to an FLASH_EraseInitTypeDef 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>

### 22.2.5 HAL\_FLASHEx\_Erase\_IT

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_Erase_IT</b> (FLASH_EraseInitTypeDef * pEraseInit)
Function Description	Perform a mass erase or erase the specified FLASH memory sectors with interrupt enabled.
Parameters	<ul style="list-style-type: none"> <li>• <b>pEraseInit:</b> pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL Status</li> </ul>

### 22.2.6 HAL\_FLASHEx\_OBProgram

Function Name	<b>HAL_StatusTypeDef HAL_FLASHEx_OBProgram</b> (FLASH_OBProgramInitTypeDef * pOBInit)
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>

### 22.2.7 HAL\_FLASHEx\_OBGetConfig

Function Name	<b>void HAL_FLASHEx_OBGetConfig</b> <b>(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>

## 22.3 FLASHEx Firmware driver defines

### 22.3.1 FLASHEx

#### **FLASH Boot Address**

OB_BOOTADDR_ITCM_RAM	Boot from ITCM RAM (0x00000000)
OB_BOOTADDR_SYSTEM	Boot from System memory bootloader (0x00100000)
OB_BOOTADDR_ITCM_FLASH	Boot from Flash on ITCM interface (0x00200000)
OB_BOOTADDR_AXIM_FLASH	Boot from Flash on AXIM interface (0x08000000)
OB_BOOTADDR_DTCM_RAM	Boot from DTCM RAM (0x20000000)
OB_BOOTADDR_SRAM1	Boot from SRAM1 (0x20010000)
OB_BOOTADDR_SRAM2	Boot from SRAM2 (0x2004C000)

#### **FLASH BOR Reset Level**

OB_BOR_LEVEL3	Supply voltage ranges from 2.70 to 3.60 V
OB_BOR_LEVEL2	Supply voltage ranges from 2.40 to 2.70 V
OB_BOR_LEVEL1	Supply voltage ranges from 2.10 to 2.40 V
OB_BOR_OFF	Supply voltage ranges from 1.62 to 2.10 V

#### **FLASH Private macros to check input parameters**

IS\_FLASH\_TYPEERASE  
 IS\_VOLTAGERANGE  
 IS\_WRPSTATE  
 IS\_OPTIONBYTE  
 IS\_OB\_BOOT\_ADDRESS  
 IS\_OB\_RDP\_LEVEL  
 IS\_OB\_WWDG\_SOURCE  
 IS\_OB\_IWDG\_SOURCE  
 IS\_OB\_STOP\_SOURCE  
 IS\_OB\_STDBY\_SOURCE  
 IS\_OB\_IWDG\_STOP\_FREEZE  
 IS\_OB\_IWDG\_STDBY\_FREEZE  
 IS\_OB\_BOR\_LEVEL  
 IS\_FLASH\_LATENCY

IS\_FLASH\_SECTOR

IS\_FLASH\_ADDRESS

IS\_FLASH\_NBSECTORS

IS\_OB\_WRP\_SECTOR

**FLASH Mass Erase bit**

FLASH\_MER\_BIT      MER bit to clear

**FLASH Option Bytes IWatchdog**

OB\_IWDG\_SW          Software IWDG selected

OB\_IWDG\_HW          Hardware IWDG selected

**FLASH IWDG Counter Freeze in STANDBY**

OB\_IWDG\_STDBY\_FREEZE   Freeze IWDG counter in STANDBY mode

OB\_IWDG\_STDBY\_ACTIVE   IWDG counter active in STANDBY mode

**FLASH IWDG Counter Freeze in STOP**

OB\_IWDG\_STOP\_FREEZE   Freeze IWDG counter in STOP mode

OB\_IWDG\_STOP\_ACTIVE   IWDG counter active in STOP mode

**FLASH Option Bytes nRST\_STDBY**

OB\_STDBY\_NO\_RST      No reset generated when entering in STANDBY

OB\_STDBY\_RST          Reset generated when entering in STANDBY

**FLASH Option Bytes nRST\_STOP**

OB\_STOP\_NO\_RST        No reset generated when entering in STOP

OB\_STOP\_RST           Reset generated when entering in STOP

**FLASH Option Bytes Read Protection**

OB\_RDP\_LEVEL\_0

OB\_RDP\_LEVEL\_1

**FLASH Option Bytes Write Protection**

OB\_WRP\_SECTOR\_0      Write protection of Sector0

OB\_WRP\_SECTOR\_1      Write protection of Sector1

OB\_WRP\_SECTOR\_2      Write protection of Sector2

OB\_WRP\_SECTOR\_3      Write protection of Sector3

OB\_WRP\_SECTOR\_4      Write protection of Sector4

OB\_WRP\_SECTOR\_5      Write protection of Sector5

OB\_WRP\_SECTOR\_6      Write protection of Sector6

OB\_WRP\_SECTOR\_7      Write protection of Sector7

OB\_WRP\_SECTOR\_All    Write protection of all Sectors

**FLASH Option Bytes WWatchdog**

OB\_WWDG\_SW          Software WWDG selected

OB\_WWDG\_HW      Hardware WWDG selected

**FLASH Option Type**

OPTIONBYTE\_WRP      WRP option byte configuration  
OPTIONBYTE\_RDP      RDP option byte configuration  
OPTIONBYTE\_USER      USER option byte configuration  
OPTIONBYTE\_BOR      BOR option byte configuration  
OPTIONBYTE\_BOOTADDR\_0      Boot 0 Address configuration  
OPTIONBYTE\_BOOTADDR\_1      Boot 1 Address configuration

**FLASH Private Constants**

SECTOR\_MASK  
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 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 Vpp

**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

## 23 HAL GPIO Generic Driver

### 23.1 GPIO Firmware driver registers structures

#### 23.1.1 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](#)

### 23.2 GPIO Firmware driver API description

#### 23.2.1 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.

### 23.2.2 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.

### 23.2.3 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\(\)\*\*](#)

## 23.2.4 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\(\)](#)

## 23.2.5 HAL\_GPIO\_Init

Function Name	<b>void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)</b>
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..K) to select the GPIO peripheral.</li> <li>• <b>GPIO_Init:</b> pointer to a GPIO_InitTypeDef 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>

## 23.2.6 HAL\_GPIO\_DeInit

Function Name	<b>void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)</b>
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..K) to select the GPIO peripheral.</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> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 23.2.7 HAL\_GPIO\_ReadPin

Function Name	<b>GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Reads the specified input port pin.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..K) to select the GPIO peripheral.</li> <li>• <b>GPIO_Pin:</b> specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• The input port pin value.</li> </ul>

## 23.2.8 HAL\_GPIO\_WritePin

Function Name	<b>void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)</b>
Function Description	Sets or clears the selected data port bit.



Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..K) to select the GPIO peripheral.</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 pinGPIO_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>

### 23.2.9 HAL\_GPIO\_TogglePin

Function Name	<b>void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Toggles the specified GPIO pins.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> Where x can be (A..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>

### 23.2.10 HAL\_GPIO\_LockPin

Function Name	<b>HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</b>
Function Description	Locks GPIO Pins configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIOx:</b> where x can be (A..F) to select the GPIO peripheral for STM32F7 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_AFR1 and GPIOx_AFR2.</li> <li>• The configuration of the locked GPIO pins can no longer be modified until the next reset.</li> </ul>

### 23.2.11 HAL\_GPIO\_EXTI\_IRQHandler

Function Name	<b>void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)</b>
Function Description	This function handles EXTI interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>GPIO_Pin:</b> Specifies the pins connected EXTI line</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 23.2.12 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>

## 23.3 GPIO Firmware driver defines

### 23.3.1 GPIO

#### *GPIO Alternate Function Selection*

GPIO\_AF0\_RTC\_50Hz

GPIO\_AF0\_MCO

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\_AF3\_LPTIM1

GPIO\_AF3\_CEC

GPIO\_AF4\_I2C1

GPIO\_AF4\_I2C2

GPIO\_AF4\_I2C3

GPIO\_AF4\_I2C4

GPIO\_AF4\_CEC

GPIO\_AF5\_SPI1

GPIO\_AF5\_SPI2

GPIO\_AF5\_SPI3

GPIO\_AF5\_SPI4

GPIO\_AF5\_SPI5

GPIO\_AF5\_SPI6

GPIO\_AF6\_SPI3

GPIO\_AF6\_SAI1

GPIO\_AF7\_USART1  
GPIO\_AF7\_USART2  
GPIO\_AF7\_USART3  
GPIO\_AF7\_UART5  
GPIO\_AF7\_SPDIFRX  
GPIO\_AF7\_SPI2  
GPIO\_AF7\_SPI3  
GPIO\_AF8\_UART4  
GPIO\_AF8\_UART5  
GPIO\_AF8\_USART6  
GPIO\_AF8\_UART7  
GPIO\_AF8\_UART8  
GPIO\_AF8\_SPDIFRX  
GPIO\_AF8\_SAI2  
GPIO\_AF9\_CAN1  
GPIO\_AF9\_CAN2  
GPIO\_AF9\_TIM12  
GPIO\_AF9\_TIM13  
GPIO\_AF9\_TIM14  
GPIO\_AF9\_QUADSPI  
GPIO\_AF9\_LTDC  
GPIO\_AF10\_OTG\_FS  
GPIO\_AF10\_OTG\_HS  
GPIO\_AF10\_QUADSPI  
GPIO\_AF10\_SAI2  
GPIO\_AF11\_ETH  
GPIO\_AF12\_FMC  
GPIO\_AF12\_OTG\_HS\_FS  
GPIO\_AF12\_SDMMC1  
GPIO\_AF13\_DCMI  
GPIO\_AF14\_LTDC  
GPIO\_AF15\_EVENTOUT

**GPIO Exported Macros**

\_\_HAL\_GPIO\_EXTI\_GET\_FLAG

**Description:**

- Checks whether the specified EXTI line flag is set or not.

**Parameters:**

`__HAL_GPIO_EXTI_CLEAR_FLAG`

- `__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).

**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\_LOW       Low speed

GPIO\_SPEED\_MEDIUM   Medium speed

GPIO\_SPEED\_FAST      Fast speed

GPIO\_SPEED\_HIGH      High speed

## 24 HAL GPIO Extension Driver

### 24.1 GPIOEx Firmware driver defines

#### 24.1.1 GPIOEx

##### *GPIO Get Port Index*

GPIO\_GET\_INDEX

##### *GPIO Check Alternate Function*

IS\_GPIO\_AF

##### *GPIO Private Constants*

GPIOA\_PIN\_AVAILABLE

GPIOB\_PIN\_AVAILABLE

GPIOC\_PIN\_AVAILABLE

GPIOD\_PIN\_AVAILABLE

GPIOE\_PIN\_AVAILABLE

GPIOF\_PIN\_AVAILABLE

GPIOG\_PIN\_AVAILABLE

GPIOI\_PIN\_AVAILABLE

GPIOJ\_PIN\_AVAILABLE

GPIOH\_PIN\_AVAILABLE

GPIOK\_PIN\_AVAILABLE

##### *GPIO Private Macros*

IS\_GPIO\_PIN\_AVAILABLE

## 25 HAL HASH Generic Driver

### 25.1 HASH Firmware driver registers structures

#### 25.1.1 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

#### 25.1.2 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\_STATTypeDef* **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 input buffer



- **`__IO uint32_t HASH_HandleTypeDef::HashBuffSize`**  
Size of buffer to be processed
- **`__IO uint32_t HASH_HandleTypeDef::HashInCount`**  
Counter of inputted data
- **`__IO uint32_t HASH_HandleTypeDef::HashITCounter`**  
Counter of issued interrupts
- **`HAL_StatusTypeDef HASH_HandleTypeDef::Status`**  
HASH peripheral status
- **`HAL_HASHPhaseTypeDef HASH_HandleTypeDef::Phase`**  
HASH peripheral phase
- **`DMA_HandleTypeDef* HASH_HandleTypeDef::hdmain`**  
HASH In DMA handle parameters
- **`HAL_LockTypeDef HASH_HandleTypeDef::Lock`**  
HASH locking object
- **`__IO HAL_HASH_STATTypeDef HASH_HandleTypeDef::State`**  
HASH peripheral state

## 25.2 HASH Firmware driver API description

### 25.2.1 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\_DeInit() to deinitialize the HASH peripheral.

### 25.2.2 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\(\)\*](#)

### 25.2.3 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\(\)\*](#)

### 25.2.4 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\(\)\*](#)

### 25.2.5 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\(\)\*](#)

### 25.2.6 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\(\)\*](#)

### 25.2.7 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\(\)\*](#)

### 25.2.8 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\(\)\*](#)

### 25.2.9 HAL\_HASH\_MD5\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Start</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
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>

### 25.2.10 HAL\_HASH\_MD5\_Accumulate

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Accumulate</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
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>

### 25.2.11 HAL\_HASH\_SHA1\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
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
- HAL status

### 25.2.12 HAL\_HASH\_SHA1\_Accumulate

- Function Name** **HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Accumulate (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**
- Function Description** Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
- Parameters**
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
  - **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
- Return values**
- HAL status

### 25.2.13 HAL\_HASH\_MD5\_Start\_IT

- Function Name** **HAL\_StatusTypeDef HAL\_HASH\_MD5\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**
- Function Description** Initializes the HASH peripheral in MD5 mode then processes pInBuffer.
- Parameters**
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
  - **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
  - **pOutBuffer:** Pointer to the computed digest. Its size must be 16 bytes.
- Return values**
- HAL status

### 25.2.14 HAL\_HASH\_SHA1\_Start\_IT

- Function Name** **HAL\_StatusTypeDef HAL\_HASH\_SHA1\_Start\_IT (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer)**
- Function Description** Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
- Parameters**
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
  - **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
  - **pOutBuffer:** Pointer to the computed digest. Its size must be 20 bytes.
- Return values**
- HAL status

**25.2.15 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>

**25.2.16 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>

**25.2.17 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>

**25.2.18 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>

### 25.2.19 HAL\_HASH\_MD5\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Finish</b> (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)
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>

### 25.2.20 HAL\_HASH\_SHA1\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
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>

### 25.2.21 HAL\_HASH\_SHA1\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Finish</b> (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)
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>

### 25.2.22 HAL\_HASH\_SHA1\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start_IT</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)
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>

### 25.2.23 HAL\_HASH\_MD5\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Start_IT</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)
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>

### 25.2.24 HAL\_HMAC\_MD5\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_MD5_Start</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
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>

### 25.2.25 HAL\_HMAC\_SHA1\_Start



Function Name	<b>HAL_StatusTypeDef HAL_HMAC_SHA1_Start</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
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>

### 25.2.26 HAL\_HASH\_SHA1\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
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>

### 25.2.27 HAL\_HASH\_SHA1\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_SHA1_Finish</b> (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)
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>

### 25.2.28 HAL\_HASH\_MD5\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
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>

### 25.2.29 HAL\_HASH\_MD5\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASH_MD5_Finish</b> (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)
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>

### 25.2.30 HAL\_HMAC\_MD5\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_MD5_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
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>

### 25.2.31 HAL\_HMAC\_SHA1\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HMAC_SHA1_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
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>

**25.2.32 HAL\_HASH\_GetState**

Function Name	<b>HAL_HASH_STATTypeDef 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>

**25.2.33 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>

**25.2.34 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>

**25.2.35 HAL\_HASH\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_HASH_DeInit (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	<ul style="list-style-type: none"> <li>This API must be called before starting a new processing.</li> </ul>

**25.2.36 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	<ul style="list-style-type: none"> <li>None</li> </ul>

**25.2.37 HAL\_HASH\_MspDeInit**

Function Name	<b>void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)</b>
Function Description	DeInitializes HASH MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 25.2.38 HAL\_HASH\_InCpltCallback

Function Name	<b>void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)</b>
Function Description	Input data transfer complete callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 25.2.39 HAL\_HASH\_ErrorCallback

Function Name	<b>void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)</b>
Function Description	Data transfer Error callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 25.2.40 HAL\_HASH\_DgstCpltCallback

Function Name	<b>void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)</b>
Function Description	Digest computation complete callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This callback is not relevant with DMA.</li> </ul>

### 25.2.41 HAL\_HASH\_GetState

Function Name	<b>HAL_HASH_STATTypeDef 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>

### 25.2.42 HAL\_HASH\_MspltInit

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

### 25.2.43 HAL\_HASH\_MspDeInit

Function Name	<b>void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)</b>
Function Description	DeInitializes HASH MSP.
Parameters	<ul style="list-style-type: none"> <li><b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 25.2.44 HAL\_HASH\_InCpltCallback

Function Name	<b>void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)</b>
Function Description	Input data transfer complete callback.
Parameters	<ul style="list-style-type: none"> <li><b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

### 25.2.45 HAL\_HASH\_DgstCpltCallback

Function Name	<b>void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)</b>
Function Description	Digest computation complete callback.
Parameters	<ul style="list-style-type: none"> <li><b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This callback is not relevant with DMA.</li> </ul>

### 25.2.46 HAL\_HASH\_ErrorCallback

Function Name	<b>void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)</b>
Function Description	Data transfer Error callback.
Parameters	<ul style="list-style-type: none"> <li><b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

## 25.3 HASH Firmware driver defines

### 25.3.1 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\_SHA224 HASH function is SHA224  
 HASH\_ALGOSELECTION\_SHA256 HASH function is SHA256  
 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\_SET\_MDMAT****Description:**

- Enable the multiple DMA mode.

**Return value:**

- None

**\_\_HAL\_HASH\_RESET\_MDMAT****Description:**

- Disable the multiple DMA mode.

**Return value:**

- None

**\_\_HAL\_HASH\_START\_DIGEST****Description:**

- Start the digest computation.

**Return value:**

- None

**\_\_HAL\_HASH\_SET\_NBVALIDBITS****Description:**

- Set the number of valid bits in 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



## 26 HAL HASH Extension Driver

### 26.1 HASHEX Firmware driver API description

#### 26.1.1 How to use this driver

The HASH HAL driver can be used as follows:

1. Initialize the HASH low level resources by implementing the HAL\_HASH\_MspInit():
  - a. Enable the HASH interface clock using `__HAL_RCC_HASH_CLK_ENABLE()`
  - b. In case of using processing APIs based on interrupts (e.g. `HAL_HMACEx_SHA224_Start()`)
    - Configure the HASH interrupt priority using `HAL_NVIC_SetPriority()`
    - Enable the HASH IRQ handler using `HAL_NVIC_EnableIRQ()`
    - In HASH IRQ handler, call `HAL_HASH_IRQHandler()`
  - c. In case of using DMA to control data transfer (e.g. `HAL_HMACEx_SHA224_Start_DMA()`)
    - Enable the DMAx interface clock using `__DMAx_CLK_ENABLE()`
    - Configure and enable one DMA stream one for managing data transfer from memory to peripheral (input stream). Managing data transfer from peripheral to memory can be performed only using CPU
    - Associate the initialized DMA handle to the HASH DMA handle using `__HAL_LINKDMA()`
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Stream: `HAL_NVIC_SetPriority()` and `HAL_NVIC_EnableIRQ()`
2. Initialize the HASH HAL using `HAL_HASH_Init()`. This function configures mainly:
  - a. The data type: 1-bit, 8-bit, 16-bit and 32-bit.
  - b. For HMAC, the encryption key.
  - c. For HMAC, the key size used for encryption.
3. Three processing functions are available:
  - a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished e.g. `HAL_HASHEX_SHA224_Start()`
  - b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt e.g. `HAL_HASHEX_SHA224_Start_IT()`
  - c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA e.g. `HAL_HASHEX_SHA224_Start_DMA()`
4. When the processing function is called at first time after `HAL_HASH_Init()` the HASH peripheral is initialized and processes the buffer in input. After that, the digest computation is started. When processing multi-buffer use the accumulate function to write the data in the peripheral without starting the digest computation. In last buffer use the start function to input the last buffer and start the digest computation.
  - a. e.g. `HAL_HASHEX_SHA224_Accumulate()` : write 1st data buffer in the peripheral without starting the digest computation
  - b. write (n-1)th data buffer in the peripheral without starting the digest computation
  - c. `HAL_HASHEX_SHA224_Start()` : write (n)th data buffer in the peripheral and start the digest computation
5. In HMAC mode, there is no Accumulate API. Only Start API is available.
6. In case of using DMA, call the DMA start processing e.g. `HAL_HASHEX_SHA224_Start_DMA()`. After that, call the finish function in order to get the digest value e.g. `HAL_HASHEX_SHA224_Finish()`

7. Call HAL\_HASH\_DeInit() to deinitialize the HASH peripheral.

### 26.1.2 HASH processing using polling mode functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- SHA224
- SHA256

This section contains the following APIs:

- [\*HAL\\_HASHEx\\_SHA224\\_Start\(\)\*](#)
- [\*HAL\\_HASHEx\\_SHA256\\_Start\(\)\*](#)
- [\*HAL\\_HASHEx\\_SHA224\\_Accumulate\(\)\*](#)
- [\*HAL\\_HASHEx\\_SHA256\\_Accumulate\(\)\*](#)

### 26.1.3 HMAC processing using polling mode functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- SHA224
- SHA256

This section contains the following APIs:

- [\*HAL\\_HMACEx\\_SHA224\\_Start\(\)\*](#)
- [\*HAL\\_HMACEx\\_SHA256\\_Start\(\)\*](#)

### 26.1.4 HASH processing using interrupt functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- SHA224
- SHA256

This section contains the following APIs:

- [\*HAL\\_HASHEx\\_SHA224\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_HASHEx\\_SHA256\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_HASHEx\\_IRQHandler\(\)\*](#)

### 26.1.5 HASH processing using DMA functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- SHA224
- SHA256

This section contains the following APIs:

- [\*HAL\\_HASHEx\\_SHA224\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HASHEx\\_SHA224\\_Finish\(\)\*](#)
- [\*HAL\\_HASHEx\\_SHA256\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HASHEx\\_SHA256\\_Finish\(\)\*](#)

## 26.1.6 HMAC processing using DMA functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- SHA224
- SHA256

This section contains the following APIs:

- [\*HAL\\_HMACEx\\_SHA224\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_HMACEx\\_SHA256\\_Start\\_DMA\(\)\*](#)

## 26.1.7 HAL\_HASHEx\_SHA224\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b>: Pointer to the computed digest. Its size must be 28 bytes.</li> <li>• <b>Timeout</b>: Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 26.1.8 HAL\_HASHEx\_SHA256\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b>: Pointer to the computed digest. Its size must be 32 bytes.</li> <li>• <b>Timeout</b>: Specify Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

## 26.1.9 HAL\_HASHEx\_SHA224\_Accumulate

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Accumulate</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
---------------	---

Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer:</b> Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size:</b> Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.10 HAL\_HASHEx\_SHA256\_Accumulate

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</b>
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer:</b> Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size:</b> Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.11 HAL\_HMACEx\_SHA224\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HMACEx_SHA224_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 SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer:</b> Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size:</b> Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer:</b> Pointer to the computed digest. Its size must be 20 bytes.</li> <li>• <b>Timeout:</b> Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.12 HAL\_HMACEx\_SHA256\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HMACEx_SHA256_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 SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>

- **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
  - **pOutBuffer:** Pointer to the computed digest. Its size must be 20 bytes.
  - **Timeout:** Timeout value
- Return values
- HAL status

### 26.1.13 HAL\_HASHEx\_SHA224\_Start\_IT

- Function Name** `HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)`
- Function Description** Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
- Parameters**
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
  - **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
  - **pOutBuffer:** Pointer to the computed digest. Its size must be 20 bytes.
- Return values
- HAL status

### 26.1.14 HAL\_HASHEx\_SHA256\_Start\_IT

- Function Name** `HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)`
- Function Description** Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
- Parameters**
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
  - **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
  - **pOutBuffer:** Pointer to the computed digest. Its size must be 20 bytes.
- Return values
- HAL status

### 26.1.15 HAL\_HASHEx\_IRQHandler

- Function Name** `void HAL_HASHEx_IRQHandler (HASH_HandleTypeDef * hhash)`
- Function Description** This function handles HASH interrupt request.
- Parameters**
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
- Return values
- None

**26.1.16 HAL\_HASHEx\_SHA224\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
Function Description	Initializes the HASH peripheral in SHA224 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**26.1.17 HAL\_HASHEx\_SHA224\_Finish**

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish</b> (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)
Function Description	Returns the computed digest in SHA224.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pOutBuffer</b>: Pointer to the computed digest. Its size must be 28 bytes.</li> <li>• <b>Timeout</b>: Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**26.1.18 HAL\_HASHEx\_SHA256\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
Function Description	Initializes the HASH peripheral in SHA256 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**26.1.19 HAL\_HASHEx\_SHA256\_Finish**

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish</b> (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)
Function Description	Returns the computed digest in SHA256.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> </ul>

- **pOutBuffer:** Pointer to the computed digest. Its size must be 32 bytes.
  - **Timeout:** Timeout value
- Return values
- HAL status

### 26.1.20 HAL\_HMACEx\_SHA224\_Start\_DMA

- Function Name **HAL\_StatusTypeDef HAL\_HMACEx\_SHA224\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**
- Function Description Initializes the HASH peripheral in HMAC SHA224 mode then enables DMA to control data transfer.
- Parameters
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
  - **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
- Return values
- HAL status

### 26.1.21 HAL\_HMACEx\_SHA256\_Start\_DMA

- Function Name **HAL\_StatusTypeDef HAL\_HMACEx\_SHA256\_Start\_DMA (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)**
- Function Description Initializes the HASH peripheral in HMAC SHA256 mode then enables DMA to control data transfer.
- Parameters
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
  - **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
- Return values
- HAL status

### 26.1.22 HAL\_HASHEx\_SHA224\_Start

- Function Name **HAL\_StatusTypeDef HAL\_HASHEx\_SHA224\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)**
- Function Description Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
- Parameters
- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
  - **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
  - **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
  - **pOutBuffer:** Pointer to the computed digest. Its size must be 28 bytes.
  - **Timeout:** Specify Timeout value

Return values

- HAL status

### 26.1.23 HAL\_HASHEx\_SHA256\_Start

**Function Name** HAL\_StatusTypeDef HAL\_HASHEx\_SHA256\_Start (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size, uint8\_t \* pOutBuffer, uint32\_t Timeout)

**Function Description** Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.

**Parameters**

- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
- **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
- **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.
- **pOutBuffer:** Pointer to the computed digest. Its size must be 32 bytes.
- **Timeout:** Specify Timeout value

Return values

- HAL status

### 26.1.24 HAL\_HASHEx\_SHA224\_Accumulate

**Function Name** HAL\_StatusTypeDef HAL\_HASHEx\_SHA224\_Accumulate (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)

**Function Description** Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.

**Parameters**

- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
- **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
- **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.

Return values

- HAL status

### 26.1.25 HAL\_HASHEx\_SHA256\_Accumulate

**Function Name** HAL\_StatusTypeDef HAL\_HASHEx\_SHA256\_Accumulate (HASH\_HandleTypeDef \* hhash, uint8\_t \* pInBuffer, uint32\_t Size)

**Function Description** Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.

**Parameters**

- **hhash:** pointer to a HASH\_HandleTypeDef structure that contains the configuration information for HASH module
- **pInBuffer:** Pointer to the input buffer (buffer to be hashed).
- **Size:** Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.

Return values

- HAL status

### 26.1.26 HAL\_HMACEx\_SHA224\_Start



Function Name	<b>HAL_StatusTypeDef HAL_HMACEx_SHA224_Start</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
Function Description	Initializes the HASH peripheral in HMAC SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b>: Pointer to the computed digest. Its size must be 20 bytes.</li> <li>• <b>Timeout</b>: Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.27 HAL\_HMACEx\_SHA256\_Start

Function Name	<b>HAL_StatusTypeDef HAL_HMACEx_SHA256_Start</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)
Function Description	Initializes the HASH peripheral in HMAC SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b>: Pointer to the computed digest. Its size must be 20 bytes.</li> <li>• <b>Timeout</b>: Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.28 HAL\_HASHEx\_SHA224\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b>: Pointer to the computed digest. Its size must be 20 bytes.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.29 HAL\_HASHEx\_SHA256\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_IT</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> <li>• <b>pOutBuffer</b>: Pointer to the computed digest. Its size must be 20 bytes.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.30 HAL\_HASHEx\_SHA224\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
Function Description	Initializes the HASH peripheral in SHA224 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer</b>: Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size</b>: Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.31 HAL\_HASHEx\_SHA224\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish</b> (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)
Function Description	Returns the computed digest in SHA224.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash</b>: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pOutBuffer</b>: Pointer to the computed digest. Its size must be 28 bytes.</li> <li>• <b>Timeout</b>: Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.32 HAL\_HASHEx\_SHA256\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA</b> (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)
Function Description	Initializes the HASH peripheral in SHA256 mode then enables DMA to control data transfer.

Parameters	<ul style="list-style-type: none"> <li>• <b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer:</b> Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size:</b> Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.33 HAL\_HASHEx\_SHA256\_Finish

Function Name	<b>HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)</b>
Function Description	Returns the computed digest in SHA256.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pOutBuffer:</b> Pointer to the computed digest. Its size must be 32 bytes.</li> <li>• <b>Timeout:</b> Timeout value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.34 HAL\_HMACEx\_SHA224\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</b>
Function Description	Initializes the HASH peripheral in HMAC SHA224 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer:</b> Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size:</b> Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 26.1.35 HAL\_HMACEx\_SHA256\_Start\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</b>
Function Description	Initializes the HASH peripheral in HMAC SHA256 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhash:</b> pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module</li> <li>• <b>pInBuffer:</b> Pointer to the input buffer (buffer to be hashed).</li> <li>• <b>Size:</b> Length of the input buffer in bytes. If the Size is not multiple of 64 bytes, the padding is managed by hardware.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

---

**26.1.36 HAL\_HASHEx\_IRQHandler**

Function Name	<b>void HAL_HASHEx_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>

## 27 HAL HCD Generic Driver

### 27.1 HCD Firmware driver registers structures

#### 27.1.1 HCD\_HandleTypeDef

##### Data Fields

- ***HCD\_TypeDef \* Instance***
- ***HCD\_InitTypeDef Init***
- ***HCD\_HCTTypeDef 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\_HCTTypeDef 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

### 27.2 HCD Firmware driver API description

#### 27.2.1 How to use this driver

1. Declare a HCD\_HandleTypeDef handle structure, for example: HCD\_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL\_HCD\_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL\_HCD\_MspInit() API:
  - a. Enable the HCD/USB Low Level interface clock using the following macros
    - \_\_OTGFS-OTG\_CLK\_ENABLE() or \_\_OTGHS-OTG\_CLK\_ENABLE()
    - \_\_OTGHSULPI\_CLK\_ENABLE() For High Speed Mode
  - b. Initialize the related GPIO clocks
  - c. Configure HCD pin-out
  - d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
  - a. hhcd.pData = phost;
6. Enable HCD transmission and reception:

- a. `HAL_HCD_Start();`

## 27.2.2 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\_DeInit\(\)`](#)
- [`HAL\_HCD\_MspInit\(\)`](#)
- [`HAL\_HCD\_MspDeInit\(\)`](#)

## 27.2.3 IO operation functions

This section contains the following APIs:

- [`HAL\_HCD\_HC\_SubmitRequest\(\)`](#)
- [`HAL\_HCD\_IRQHandler\(\)`](#)
- [`HAL\_HCD\_SOF\_Callback\(\)`](#)
- [`HAL\_HCD\_Connect\_Callback\(\)`](#)
- [`HAL\_HCD\_Disconnect\_Callback\(\)`](#)
- [`HAL\_HCD\_HC\_NotifyURBChange\_Callback\(\)`](#)

## 27.2.4 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\(\)`](#)

## 27.2.5 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\(\)`](#)

## 27.2.6 HAL\_HCD\_Init

Function Name	<b>HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)</b>
Function Description	Initialize the host driver.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**27.2.7 HAL\_HCD\_HC\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)</b>
Function Description	Initialize a host channel.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd:</b> HCD handle</li> <li>• <b>ch_num:</b> Channel number. This parameter can be a value from 1 to 15</li> <li>• <b>epnum:</b> Endpoint number. This parameter can be a value from 1 to 15</li> <li>• <b>dev_address:</b> : Current device address This parameter can be a value from 0 to 255</li> <li>• <b>speed:</b> Current device speed. This parameter can be one of these values: HCD_SPEED_HIGH: High speed mode, HCD_SPEED_FULL: Full speed mode, HCD_SPEED_LOW: Low speed mode</li> <li>• <b>ep_type:</b> Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type, EP_TYPE_ISOC: Isochronous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type</li> <li>• <b>mps:</b> Max Packet Size. This parameter can be a value from 0 to 32K</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**27.2.8 HAL\_HCD\_HC\_Halt**

Function Name	<b>HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * 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>

**27.2.9 HAL\_HCD\_DeInit**

Function Name	<b>HAL_StatusTypeDef HAL_HCD_DeInit (HCD_HandleTypeDef * 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>

**27.2.10 HAL\_HCD\_Msplnit**

Function Name	<b>void HAL_HCD_Msplnit (HCD_HandleTypeDef * hhcd)</b>
Function Description	Initializes the HCD MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd:</b> HCD handle</li> </ul>

Return values • None

## 27.2.11 HAL\_HCD\_MspDeInit

Function Name **void HAL\_HCD\_MspDeInit (HCD\_HandleTypeDef \* hhcd)**

Function Description DeInitializes HCD MSP.

Parameters • **hhcd**: HCD handle

Return values • None

## 27.2.12 HAL\_HCD\_HC\_SubmitRequest

Function Name **HAL\_StatusTypeDef HAL\_HCD\_HC\_SubmitRequest (HCD\_HandleTypeDef \* hhcd, uint8\_t ch\_num, uint8\_t direction, uint8\_t ep\_type, uint8\_t token, uint8\_t \* pbuff, uint16\_t length, uint8\_t do\_ping)**

Function Description Submit a new URB for processing.

Parameters

- **hhcd**: HCD handle
- **ch\_num**: Channel number. This parameter can be a value from 1 to 15
- **direction**: Channel number. This parameter can be one of these values: 0 : Output / 1 : Input
- **ep\_type**: Endpoint Type. This parameter can be one of these values: EP\_TYPE\_CTRL: Control type/ EP\_TYPE\_ISOC: Isochronous type/ EP\_TYPE\_BULK: Bulk type/ EP\_TYPE\_INTR: Interrupt type/
- **token**: Endpoint Type. This parameter can be one of these values: 0: HC\_PID\_SETUP / 1: HC\_PID\_DATA1
- **pbuff**: pointer to URB data
- **length**: Length of URB data
- **do\_ping**: activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active

Return values • HAL status

## 27.2.13 HAL\_HCD\_IRQHandler

Function Name **void HAL\_HCD\_IRQHandler (HCD\_HandleTypeDef \* hhcd)**

Function Description This function handles HCD interrupt request.

Parameters • **hhcd**: HCD handle

Return values • None

## 27.2.14 HAL\_HCD\_SOF\_Callback

Function Name **void HAL\_HCD\_SOF\_Callback (HCD\_HandleTypeDef \* hhcd)**

Function Description SOF callback.

Parameters • **hhcd**: HCD handle

Return values • None



**27.2.15 HAL\_HCD\_Connect\_Callback**

Function Name	<b>void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)</b>
Function Description	Connexion Event callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hhcd</b>: HCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**27.2.16 HAL\_HCD\_Disconnect\_Callback**

Function Name	<b>void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)</b>
Function Description	Disconnexion 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>

**27.2.17 HAL\_HCD\_HC\_NotifyURBChange\_Callback**

Function Name	<b>void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * 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>

**27.2.18 HAL\_HCD\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * 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>

**27.2.19 HAL\_HCD\_Stop**

Function Name	<b>HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * 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>

**27.2.20 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>

**27.2.21 HAL\_HCD\_GetState**

Function Name	<b>HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef * hhcd)</b>
Function Description	Return the HCD state.
Parameters	<ul style="list-style-type: none"><li>• <b>hhcd</b>: HCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL state</li></ul>

**27.2.22 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>

**27.2.23 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>

**27.2.24 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
- Host channel state This parameter can be one of the these values: HC\_IDLE/ HC\_XFRC/ HC\_HALTED/ HC\_NYET/ HC\_NAK/ HC\_STALL/ HC\_XACTERR/ HC\_BBLERR/ HC\_DATATGLERR/

### 27.2.25 HAL\_HCD\_GetCurrentFrame

Function Name **uint32\_t HAL\_HCD\_GetCurrentFrame (HCD\_HandleTypeDef \* hhcd)**

Function Description Return the current Host frame number.

Parameters

- **hhcd**: HCD handle

Return values

- Current Host frame number

### 27.2.26 HAL\_HCD\_GetCurrentSpeed

Function Name **uint32\_t HAL\_HCD\_GetCurrentSpeed (HCD\_HandleTypeDef \* hhcd)**

Function Description Return the Host enumeration speed.

Parameters

- **hhcd**: HCD handle

Return values

- Enumeration speed

## 27.3 HCD Firmware driver defines

### 27.3.1 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

## 28 HAL I2C Generic Driver

### 28.1 I2C Firmware driver registers structures

#### 28.1.1 I2C\_InitTypeDef

##### Data Fields

- *uint32\_t Timing*
- *uint32\_t OwnAddress1*
- *uint32\_t AddressingMode*
- *uint32\_t DualAddressMode*
- *uint32\_t OwnAddress2*
- *uint32\_t OwnAddress2Masks*
- *uint32\_t GeneralCallMode*
- *uint32\_t NoStretchMode*

##### Field Documentation

- ***uint32\_t I2C\_InitTypeDef::Timing***  
Specifies the I2C\_TIMINGR\_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32\_t I2C\_InitTypeDef::OwnAddress1***  
Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32\_t I2C\_InitTypeDef::AddressingMode***  
Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of [I2C\\_addressing\\_mode](#)
- ***uint32\_t I2C\_InitTypeDef::DualAddressMode***  
Specifies if dual addressing mode is selected. This parameter can be a value of [I2C\\_dual\\_addressing\\_mode](#)
- ***uint32\_t I2C\_InitTypeDef::OwnAddress2***  
Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32\_t I2C\_InitTypeDef::OwnAddress2Masks***  
Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of [I2C\\_own\\_address2\\_masks](#)
- ***uint32\_t I2C\_InitTypeDef::GeneralCallMode***  
Specifies if general call mode is selected. This parameter can be a value of [I2C\\_general\\_call\\_addressing\\_mode](#)
- ***uint32\_t I2C\_InitTypeDef::NoStretchMode***  
Specifies if nostretch mode is selected. This parameter can be a value of [I2C\\_nostretch\\_mode](#)

#### 28.1.2 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

## 28.2 I2C Firmware driver API description

### 28.2.1 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 Clock Timing, Own Address1, Master Addressing Mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
  4. Initialize the I2C registers by calling the HAL\_I2C\_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL\_I2C\_MspInit(&hi2c) API.
  5. To check if target device is ready for communication, use the function HAL\_I2C\_IsDeviceReady()
  6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

### Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL\_I2C\_Master\_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL\_I2C\_Master\_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL\_I2C\_Slave\_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL\_I2C\_Slave\_Receive()

### Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL\_I2C\_Mem\_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL\_I2C\_Mem\_Read()

### Interrupt mode IO operation

- Transmit in master mode an amount of data in non blocking mode using HAL\_I2C\_Master\_Transmit\_IT()
- At transmission end of transfer HAL\_I2C\_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode using HAL\_I2C\_Master\_Receive\_IT()
- At reception end of transfer HAL\_I2C\_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode using HAL\_I2C\_Slave\_Transmit\_IT()

- At transmission end of transfer HAL\_I2C\_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode using HAL\_I2C\_Slave\_Receive\_IT()
- At reception end of transfer HAL\_I2C\_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_I2C\_SlaveRxCpltCallback
- In case of transfer Error, HAL\_I2C\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_I2C\_ErrorCallback

### 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

## 28.2.2 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:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
- Call the function HAL\_I2C\_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- [\*HAL\\_I2C\\_Init\(\)\*](#)
- [\*HAL\\_I2C\\_DeInit\(\)\*](#)
- [\*HAL\\_I2C\\_MspInit\(\)\*](#)
- [\*HAL\\_I2C\\_MspDeInit\(\)\*](#)

## 28.2.3 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\(\)](#)

## 28.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- [HAL\\_I2C\\_GetState\(\)](#)
- [HAL\\_I2C\\_GetError\(\)](#)

## 28.2.5 HAL\_I2C\_Init

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)</b>
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>• <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>• HAL status</li> </ul>

## 28.2.6 HAL\_I2C\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)</b>
Function Description	DeInitializes the I2C peripheral.
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>• HAL status</li> </ul>

## 28.2.7 HAL\_I2C\_MspInit

Function Name	<b>void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)</b>
Function Description	I2C MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b>: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 28.2.8 HAL\_I2C\_MspDeInit

Function Name	<b>void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)</b>
Function Description	I2C MSP DeInit.
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>• None</li> </ul>

**28.2.9 HAL\_I2C\_Master\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Transmit</b> (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in master mode an amount of data in blocking mode.
Parameters	<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> <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>

**28.2.10 HAL\_I2C\_Master\_Receive**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Master_Receive</b> (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives in master mode an amount of data in blocking mode.
Parameters	<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> <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>

**28.2.11 HAL\_I2C\_Slave\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Transmit</b> (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in slave mode an amount of data in blocking mode.
Parameters	<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> <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>

**28.2.12 HAL\_I2C\_Slave\_Receive**

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Slave_Receive</b> (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive in slave mode an amount of data in blocking mode.
Parameters	<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>

- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- HAL status

### 28.2.13 HAL\_I2C\_Master\_Transmit\_IT

**Function Name** `HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)`

**Function Description** Transmit in master mode an amount of data in no-blocking mode with Interrupt.

- Parameters**
- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
  - **DevAddress:** Target device address
  - **pData:** Pointer to data buffer
  - **Size:** Amount of data to be sent

Return values

- HAL status

### 28.2.14 HAL\_I2C\_Master\_Receive\_IT

**Function Name** `HAL_StatusTypeDef HAL_I2C_Master_Receive_IT(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)`

**Function Description** Receive in master mode an amount of data in no-blocking mode with Interrupt.

- Parameters**
- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
  - **DevAddress:** Target device address
  - **pData:** Pointer to data buffer
  - **Size:** Amount of data to be sent

Return values

- HAL status

### 28.2.15 HAL\_I2C\_Slave\_Transmit\_IT

**Function Name** `HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT(I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)`

**Function Description** Transmit in slave mode an amount of data in no-blocking mode with Interrupt.

- Parameters**
- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
  - **pData:** Pointer to data buffer
  - **Size:** Amount of data to be sent

Return values

- HAL status

### 28.2.16 HAL\_I2C\_Slave\_Receive\_IT

**Function Name** `HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT`

	<b>(I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive in slave mode an amount of data in no-blocking mode with Interrupt.
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> <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>
<b>28.2.17 HAL_I2C_Master_Transmit_DMA</b>	
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 the specified I2C.</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>
<b>28.2.18 HAL_I2C_Master_Receive_DMA</b>	
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 the specified I2C.</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>
<b>28.2.19 HAL_I2C_Slave_Transmit_DMA</b>	
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 the specified I2C.</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

### 28.2.20 HAL\_I2C\_Slave\_Receive\_DMA

**Function Name** HAL\_StatusTypeDef HAL\_I2C\_Slave\_Receive\_DMA  
(I2C\_HandleTypeDef \* hi2c, uint8\_t \* pData, uint16\_t Size)

**Function Description** Receive in slave mode an amount of data in no-blocking mode with DMA.

**Parameters**

- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

**Return values**

- HAL status

### 28.2.21 HAL\_I2C\_Mem\_Write

**Function Name** HAL\_StatusTypeDef HAL\_I2C\_Mem\_Write  
(I2C\_HandleTypeDef \* hi2c, uint16\_t DevAddress, uint16\_t MemAddress, uint16\_t MemAddSize, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)

**Function Description** Write an amount of data in blocking mode to a specific memory address.

**Parameters**

- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

**Return values**

- HAL status

### 28.2.22 HAL\_I2C\_Mem\_Read

**Function Name** HAL\_StatusTypeDef HAL\_I2C\_Mem\_Read  
(I2C\_HandleTypeDef \* hi2c, uint16\_t DevAddress, uint16\_t MemAddress, uint16\_t MemAddSize, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)

**Function Description** Read an amount of data in blocking mode from a specific memory address.

**Parameters**

- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

**Return values**

- HAL status

### 28.2.23 HAL\_I2C\_Mem\_Write\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Mem_Write_IT</b> (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
Function Description	Write an amount of data in no-blocking mode with Interrupt to a specific memory address.
Parameters	<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><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>

### 28.2.24 HAL\_I2C\_Mem\_Read\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Mem_Read_IT</b> (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
Function Description	Read an amount of data in no-blocking mode with Interrupt from a specific memory address.
Parameters	<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><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>

### 28.2.25 HAL\_I2C\_Mem\_Write\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA</b> (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
Function Description	Write an amount of data in no-blocking mode with DMA to a specific memory address.
Parameters	<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><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

- HAL status

### 28.2.26 HAL\_I2C\_Mem\_Read\_DMA

**Function Name** `HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)`

**Function Description** Reads an amount of data in no-blocking mode with DMA from a specific memory address.

**Parameters**

- **hi2c**: : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress**: Target device address
- **MemAddress**: Internal memory address
- **MemAddSize**: Size of internal memory address
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be read

Return values

- HAL status

### 28.2.27 HAL\_I2C\_IsDeviceReady

**Function Name** `HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)`

**Function Description** Checks if target device is ready for communication.

**Parameters**

- **hi2c**: : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress**: Target device address
- **Trials**: Number of trials
- **Timeout**: Timeout duration

Return values

- HAL status

**Notes**

- This function is used with Memory devices

### 28.2.28 HAL\_I2C\_EV\_IRQHandler

**Function Name** `void HAL_I2C_EV_IRQHandler(I2C_HandleTypeDef * hi2c)`

**Function Description** This function handles I2C event interrupt request.

**Parameters**

- **hi2c**: : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- None

### 28.2.29 HAL\_I2C\_ER\_IRQHandler

**Function Name** `void HAL_I2C_ER_IRQHandler(I2C_HandleTypeDef * hi2c)`

**Function Description** This function handles I2C error interrupt request.

**Parameters**

- **hi2c**: : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- None

### 28.2.30 HAL\_I2C\_MasterTxCpltCallback

Function Name **void HAL\_I2C\_MasterTxCpltCallback (I2C\_HandleTypeDef \* hi2c)**

Function Description Master Tx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- None

### 28.2.31 HAL\_I2C\_MasterRxCpltCallback

Function Name **void HAL\_I2C\_MasterRxCpltCallback (I2C\_HandleTypeDef \* hi2c)**

Function Description Master Rx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- None

### 28.2.32 HAL\_I2C\_SlaveTxCpltCallback

Function Name **void HAL\_I2C\_SlaveTxCpltCallback (I2C\_HandleTypeDef \* hi2c)**

Function Description Slave Tx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- None

### 28.2.33 HAL\_I2C\_SlaveRxCpltCallback

Function Name **void HAL\_I2C\_SlaveRxCpltCallback (I2C\_HandleTypeDef \* hi2c)**

Function Description Slave Rx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- None

### 28.2.34 HAL\_I2C\_MemTxCpltCallback

Function Name **void HAL\_I2C\_MemTxCpltCallback (I2C\_HandleTypeDef \* hi2c)**

Function Description Memory Tx Transfer completed callbacks.

Parameters

- **hi2c:** : Pointer to a I2C\_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- None

**28.2.35 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 the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**28.2.36 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 the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**28.2.37 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 the specified I2C.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

**28.2.38 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>

**28.3 I2C Firmware driver defines****28.3.1 I2C*****I2C addressing mode***

I2C\_ADDRESSINGMODE\_7BIT

I2C\_ADDRESSINGMODE\_10BIT

***I2C dual addressing mode***

I2C\_DUALADDRESS\_DISABLE

I2C\_DUALADDRESS\_ENABLE

***I2C Error Code definition***

HAL_I2C_ERROR_NONE	No error
HAL_I2C_ERROR_BERR	BERR error
HAL_I2C_ERROR_ARLO	ARLO error
HAL_I2C_ERROR_AF	ACKF error
HAL_I2C_ERROR_OVR	OVR error
HAL_I2C_ERROR_DMA	DMA transfer error
HAL_I2C_ERROR_TIMEOUT	Timeout error
HAL_I2C_ERROR_SIZE	Size Management error

**I2C Exported Macros**

`__HAL_I2C_RESET_HANDLE_STATE` **Description:**

- Reset I2C handle state.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.

**Return value:**

- None

`__HAL_I2C_ENABLE_IT`

**Description:**

- Enable the specified I2C interrupts.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable. This parameter can be one of the following values:
  - I2C\_IT\_ERRI: Errors interrupt enable
  - I2C\_IT\_TCI: Transfer complete interrupt enable
  - I2C\_IT\_STOPI: STOP detection interrupt enable
  - I2C\_IT\_NACKI: NACK received interrupt enable
  - I2C\_IT\_ADDRI: Address match interrupt enable
  - I2C\_IT\_RXI: RX interrupt enable
  - I2C\_IT\_TXI: TX interrupt enable

**Return value:**

- None

`__HAL_I2C_DISABLE_IT`

**Description:**

- Disable the specified I2C interrupts.

**Parameters:**

- `__HANDLE__`: specifies the I2C Handle.
- `__INTERRUPT__`: specifies the interrupt source to disable. This parameter can be one of the following values:

- I2C\_IT\_ERRI: Errors interrupt enable
- I2C\_IT\_TCI: Transfer complete interrupt enable
- I2C\_IT\_STOPI: STOP detection interrupt enable
- I2C\_IT\_NACKI: NACK received interrupt enable
- I2C\_IT\_ADDRI: Address match interrupt enable
- I2C\_IT\_RXI: RX interrupt enable
- I2C\_IT\_TXI: TX interrupt enable

**Return value:**

- None

**\_\_HAL\_I2C\_GET\_IT\_SOURCE****Description:**

- Checks if the specified I2C interrupt source is enabled or disabled.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.
- \_\_INTERRUPT\_\_: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  - I2C\_IT\_ERRI: Errors interrupt enable
  - I2C\_IT\_TCI: Transfer complete interrupt enable
  - I2C\_IT\_STOPI: STOP detection interrupt enable
  - I2C\_IT\_NACKI: NACK received interrupt enable
  - I2C\_IT\_ADDRI: Address match interrupt enable
  - I2C\_IT\_RXI: RX interrupt enable
  - I2C\_IT\_TXI: TX interrupt enable

**Return value:**

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

**I2C\_FLAG\_MASK****Description:**

- Checks whether the specified I2C flag is set or not.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - I2C\_FLAG\_TXE: Transmit data register empty
  - I2C\_FLAG\_TXIS: Transmit interrupt status
  - I2C\_FLAG\_RXNE: Receive data

- register not empty
- I2C\_FLAG\_ADDR: Address matched (slave mode)
- I2C\_FLAG\_AF: Acknowledge failure received flag
- I2C\_FLAG\_STOPF: STOP detection flag
- I2C\_FLAG\_TC: Transfer complete (master mode)
- I2C\_FLAG\_TCR: Transfer complete reload
- I2C\_FLAG\_BERR: Bus error
- I2C\_FLAG\_ARLO: Arbitration lost
- I2C\_FLAG\_OVR: Overrun/Underrun
- I2C\_FLAG\_PECERR: PEC error in reception
- I2C\_FLAG\_TIMEOUT: Timeout or Tlow detection flag
- I2C\_FLAG\_ALERT: SMBus alert
- I2C\_FLAG\_BUSY: Bus busy
- I2C\_FLAG\_DIR: Transfer direction (slave mode)

**Return value:**

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

\_\_HAL\_I2C\_GET\_FLAG

\_\_HAL\_I2C\_CLEAR\_FLAG

**Description:**

- Clears the I2C pending flags which are cleared by writing 1 in a specific bit.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.
- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C\_FLAG\_ADDR: Address matched (slave mode)
  - I2C\_FLAG\_AF: Acknowledge failure received flag
  - I2C\_FLAG\_STOPF: STOP detection flag
  - I2C\_FLAG\_BERR: Bus error
  - I2C\_FLAG\_ARLO: Arbitration lost
  - I2C\_FLAG\_OVR: Overrun/Underrun
  - I2C\_FLAG\_PECERR: PEC error in reception
  - I2C\_FLAG\_TIMEOUT: Timeout or Tlow detection flag
  - I2C\_FLAG\_ALERT: SMBus alert

**Return value:**

\_\_HAL\_I2C\_ENABLE

- None

**Description:**

- Enable the specified I2C peripheral.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.

**Return value:**

- None

\_\_HAL\_I2C\_DISABLE

**Description:**

- Disable the specified I2C peripheral.

**Parameters:**

- \_\_HANDLE\_\_: specifies the I2C Handle.

**Return value:**

- None

***I2C Flag definition***

I2C\_FLAG\_TXE

I2C\_FLAG\_TXIS

I2C\_FLAG\_RXNE

I2C\_FLAG\_ADDR

I2C\_FLAG\_AF

I2C\_FLAG\_STOPF

I2C\_FLAG\_TC

I2C\_FLAG\_TCR

I2C\_FLAG\_BERR

I2C\_FLAG\_ARLO

I2C\_FLAG\_OVR

I2C\_FLAG\_PECERR

I2C\_FLAG\_TIMEOUT

I2C\_FLAG\_ALERT

I2C\_FLAG\_BUSY

I2C\_FLAG\_DIR

***I2C general call addressing mode***

I2C\_GENERALCALL\_DISABLE

I2C\_GENERALCALL\_ENABLE

***I2C Interrupt configuration definition***

I2C\_IT\_ERRI

I2C\_IT\_TCI

I2C\_IT\_STOPI

I2C\_IT\_NACKI

I2C\_IT\_ADDRI

I2C\_IT\_RXI

I2C\_IT\_TXI

**I2C Memory Address Size**

I2C\_MEMADD\_SIZE\_8BIT

I2C\_MEMADD\_SIZE\_16BIT

**I2C nostretch mode**

I2C\_NOSTRETCH\_DISABLE

I2C\_NOSTRETCH\_ENABLE

**I2C own address2 masks**

I2C\_OA2\_NOMASK

I2C\_OA2\_MASK01

I2C\_OA2\_MASK02

I2C\_OA2\_MASK03

I2C\_OA2\_MASK04

I2C\_OA2\_MASK05

I2C\_OA2\_MASK06

I2C\_OA2\_MASK07

**I2C Private Constants**

TIMING\_CLEAR\_MASK

I2C\_TIMEOUT\_ADDR

I2C\_TIMEOUT\_BUSY

I2C\_TIMEOUT\_DIR

I2C\_TIMEOUT\_RXNE

I2C\_TIMEOUT\_STOPF

I2C\_TIMEOUT\_TC

I2C\_TIMEOUT\_TCR

I2C\_TIMEOUT\_TXIS

I2C\_TIMEOUT\_FLAG

**I2C Private Macros**

IS\_I2C\_ADDRESSING\_MODE

IS\_I2C\_DUAL\_ADDRESS

IS\_I2C\_OWN\_ADDRESS2\_MASK

IS\_I2C\_GENERAL\_CALL



IS\_I2C\_NO\_STRETCH  
IS\_I2C\_MEMADD\_SIZE  
IS\_TRANSFER\_MODE  
IS\_TRANSFER\_REQUEST  
I2C\_RESET\_CR2  
IS\_I2C\_OWN\_ADDRESS1  
IS\_I2C\_OWN\_ADDRESS2  
I2C\_MEM\_ADD\_MSB  
I2C\_MEM\_ADD\_LSB  
I2C\_GENERATE\_START  
IS\_I2C\_ANALOG\_FILTER  
IS\_I2C\_DIGITAL\_FILTER

***I2C ReloadEndMode definition***

I2C\_RELOAD\_MODE  
I2C\_AUTOEND\_MODE  
I2C\_SOFTEND\_MODE

***I2C StartStopMode definition***

I2C\_NO\_STARTSTOP  
I2C\_GENERATE\_STOP  
I2C\_GENERATE\_START\_READ  
I2C\_GENERATE\_START\_WRITE

## 29 HAL I2C Extension Driver

### 29.1 I2CEx Firmware driver API description

#### 29.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32L4XX devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode

#### 29.1.2 How to use this driver

This driver provides functions to:

1. Configure I2C Analog noise filter using the function `HAL_I2CEx_ConfigAnalogFilter()`
2. Configure I2C Digital noise filter using the function `HAL_I2CEx_ConfigDigitalFilter()`

#### 29.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- [`HAL\_I2CEx\_ConfigAnalogFilter\(\)`](#)
- [`HAL\_I2CEx\_ConfigDigitalFilter\(\)`](#)

#### 29.1.4 HAL\_I2CEx\_ConfigAnalogFilter

Function Name	<b>HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter(I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)</b>
Function Description	Configures I2C Analog noise filter.
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 I2Cx peripheral.</li> <li>• <b>AnalogFilter</b>: : new state of the Analog filter.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 29.1.5 HAL\_I2CEx\_ConfigDigitalFilter

Function Name	<b>HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter(I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)</b>
Function Description	Configures I2C Digital noise filter.
Parameters	<ul style="list-style-type: none"> <li>• <b>hi2c</b>: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.</li> <li>• <b>DigitalFilter</b>: : Coefficient of digital noise filter between 0x00 and 0x0F.</li> </ul>

Return values

- HAL status

## 29.2 I2CEx Firmware driver defines

### 29.2.1 I2CEx

*I2CEx Analog Filter*

I2C\_ANALOGFILTER\_ENABLE

I2C\_ANALOGFILTER\_DISABLE

## 30 HAL I2S Generic Driver

### 30.1 I2S Firmware driver registers structures

#### 30.1.1 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](#)

#### 30.1.2 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_TypeDef* I2S_HandleTypeDef::Instance`
- `I2S_InitTypeDef I2S_HandleTypeDef::Init`
- `uint16_t* I2S_HandleTypeDef::pTxBuffPtr`
- `__IO uint16_t I2S_HandleTypeDef::TxXferSize`
- `__IO uint16_t I2S_HandleTypeDef::TxXferCount`
- `uint16_t* I2S_HandleTypeDef::pRxBuffPtr`
- `__IO uint16_t I2S_HandleTypeDef::RxXferSize`
- `__IO uint16_t I2S_HandleTypeDef::RxXferCount`
- `DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx`
- `__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock`
- `__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State`
- `__IO uint32_t I2S_HandleTypeDef::ErrorCode`

## 30.2 I2S Firmware driver API description

### 30.2.1 How to use this driver

The I2S HAL driver can be used as follows:

1. Declare a `I2S_HandleTypeDef` handle structure.
2. Initialize the I2S low level resources by implement the `HAL_I2S_MspInit()` API:
  - a. Enable the SPIx interface clock.
  - b. I2S pins configuration:
    - Enable the clock for the I2S GPIOs.
    - Configure these I2S pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (`HAL_I2S_Transmit_IT()` and `HAL_I2S_Receive_IT()` APIs).
    - Configure the I2Sx interrupt priority.
    - Enable the NVIC I2S IRQ handle.
  - d. DMA Configuration if you need to use DMA process (`HAL_I2S_Transmit_DMA()` and `HAL_I2S_Receive_DMA()` APIs):
    - Declare a DMA handle structure for the Tx/Rx channel.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.
    - Configure the DMA Tx/Rx Channel.
    - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Channel.
3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using `HAL_I2S_Init()` function. The specific I2S interrupts (Transmission

complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros `__HAL_I2S_ENABLE_IT()` and `__HAL_I2S_DISABLE_IT()` inside the transmit and receive process. Make sure that either: I2S clock is configured based on SYSCCLK or External clock source is configured after setting correctly the define constant `EXTERNAL_CLOCK_VALUE` in the `stm32f3xx_hal_conf.h` file.

4. Three mode of operations are available within this driver :

### Polling mode IO operation

- Send an amount of data in blocking mode using `HAL_I2S_Transmit()`
- Receive an amount of data in blocking mode using `HAL_I2S_Receive()`

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using `HAL_I2S_Transmit_IT()`
- At transmission end of half transfer `HAL_I2S_TxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_TxHalfCpltCallback`
- At transmission end of transfer `HAL_I2S_TxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_TxCpltCallback`
- Receive an amount of data in non blocking mode using `HAL_I2S_Receive_IT()`
- At reception end of half transfer `HAL_I2S_RxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_RxHalfCpltCallback`
- At reception end of transfer `HAL_I2S_RxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_RxCpltCallback`
- In case of transfer Error, `HAL_I2S_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_I2S_ErrorCallback`

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using `HAL_I2S_Transmit_DMA()`
- At transmission end of half transfer `HAL_I2S_TxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_TxHalfCpltCallback`
- At transmission end of transfer `HAL_I2S_TxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_TxCpltCallback`
- Receive an amount of data in non blocking mode (DMA) using `HAL_I2S_Receive_DMA()`
- At reception end of half transfer `HAL_I2S_RxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_RxHalfCpltCallback`
- At reception end of transfer `HAL_I2S_RxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_RxCpltCallback`
- In case of transfer Error, `HAL_I2S_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_I2S_ErrorCallback`
- Pause the DMA Transfer using `HAL_I2S_DMABase()`
- Resume the DMA Transfer using `HAL_I2S_DMAResume()`
- Stop the DMA Transfer using `HAL_I2S_DMAStop()`

### I2S HAL driver macros list

Below the list of most used macros in I2S HAL driver.

- `__HAL_I2S_ENABLE`: Enable the specified SPI peripheral (in I2S mode)
- `__HAL_I2S_DISABLE`: Disable the specified SPI peripheral (in I2S mode)
- `__HAL_I2S_ENABLE_IT` : Enable the specified I2S interrupts
- `__HAL_I2S_DISABLE_IT` : Disable the specified I2S interrupts
- `__HAL_I2S_GET_FLAG`: Check whether the specified I2S flag is set or not



You can refer to the I2S HAL driver header file for more useful macros

### 30.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Sx peripheral in simplex mode:

- User must Implement `HAL_I2S_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function `HAL_I2S_Init()` to configure the selected device with the selected configuration:
  - Mode
  - Standard
  - Data Format
  - MCLK Output
  - Audio frequency
  - Polarity
  - Full duplex mode
- Call the function `HAL_I2S_DeInit()` to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- [\*HAL\\_I2S\\_Init\(\)\*](#)
- [\*HAL\\_I2S\\_DeInit\(\)\*](#)
- [\*HAL\\_I2S\\_MspInit\(\)\*](#)
- [\*HAL\\_I2S\\_MspDeInit\(\)\*](#)

### 30.2.3 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\(\)](#)

### 30.2.4 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\(\)](#)

### 30.2.5 HAL\_I2S\_Init

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)</b>
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>

### 30.2.6 HAL\_I2S\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)</b>
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 the configuration information for I2S module</li> </ul>



Return values

- HAL status

### 30.2.7 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

### 30.2.8 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

### 30.2.9 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).

### 30.2.10 HAL\_I2S\_Receive

Function Name **HAL\_StatusTypeDef HAL\_I2S\_Receive (I2S\_HandleTypeDef \* hi2s, uint16\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

Function Description Receive an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S\_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:

	<ul style="list-style-type: none"> <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>

### 30.2.11 HAL\_I2S\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_IT</b> (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
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>

### 30.2.12 HAL\_I2S\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_IT</b> (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
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 parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data</li> </ul>

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.

### 30.2.13 HAL\_I2S\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Transmit_DMA</b> (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
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>

### 30.2.14 HAL\_I2S\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_I2S_Receive_DMA</b> (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
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>

### 30.2.15 HAL\_I2S\_DMAPause



Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAPause (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>

### 30.2.16 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>

### 30.2.17 HAL\_I2S\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)</b>
Function Description	Stops 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>

### 30.2.18 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>HAL status</li> </ul>

### 30.2.19 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>

### 30.2.20 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>

### 30.2.21 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>

### 30.2.22 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>

### 30.2.23 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>

### 30.2.24 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>

### 30.2.25 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>

## 30.3 I2S Firmware driver defines

### 30.3.1 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\_EXTERNAL

I2S\_CLOCK\_SYSCLK

#### *I2S Data Format*

I2S\_DATAFORMAT\_16B

I2S\_DATAFORMAT\_16B\_EXTENDED

I2S\_DATAFORMAT\_24B

I2S\_DATAFORMAT\_32B

#### *I2S Error Defintion*

HAL\_I2S\_ERROR\_NONE No error

HAL\_I2S\_ERROR\_TIMEOUT Timeout error

HAL\_I2S\_ERROR\_OVR OVR error

HAL\_I2S\_ERROR\_UDR UDR error

HAL\_I2S\_ERROR\_DMA DMA transfer error

HAL\_I2S\_ERROR\_UNKNOW Unknow Error error

#### *I2S Exported Macros*

__HAL_I2S_RESET_HANDLE_STATE	<b>Description:</b>
------------------------------	---------------------

- Reset I2S handle state.

#### **Parameters:**

- \_\_HANDLE\_\_: specifies the I2S handle.

`__HAL_I2S_ENABLE`

**Return value:**

- None

**Description:**

- Enable or disable the specified SPI peripheral (in I2S mode).

**Parameters:**

- `__HANDLE__`: specifies the I2S Handle.

**Return value:**

- None

`__HAL_I2S_DISABLE`

`__HAL_I2S_ENABLE_IT`

**Description:**

- Enable or disable the specified I2S interrupts.

**Parameters:**

- `__HANDLE__`: specifies the I2S Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - `I2S_IT_TXE`: Tx buffer empty interrupt enable
  - `I2S_IT_RXNE`: RX buffer not empty interrupt enable
  - `I2S_IT_ERR`: Error interrupt enable

**Return value:**

- None

`__HAL_I2S_DISABLE_IT`

`__HAL_I2S_GET_IT_SOURCE`

**Description:**

- Checks if the specified I2S interrupt source is enabled or disabled.

**Parameters:**

- `__HANDLE__`: specifies the I2S Handle. This parameter can be I2S where x: 1, 2, or 3 to select the I2S peripheral.
- `__INTERRUPT__`: specifies the I2S interrupt source to check. This parameter can be one of the following values:
  - `I2S_IT_TXE`: Tx buffer empty interrupt enable
  - `I2S_IT_RXNE`: RX buffer not empty interrupt enable
  - `I2S_IT_ERR`: Error interrupt enable

**Return value:**

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

**\_\_HAL\_I2S\_GET\_FLAG****Description:**

- Checks whether the specified I2S flag is set or not.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - I2S\_FLAG\_RXNE: Receive buffer not empty flag
  - I2S\_FLAG\_TXE: Transmit buffer empty flag
  - I2S\_FLAG\_UDR: Underrun flag
  - I2S\_FLAG\_OVR: Overrun flag
  - I2S\_FLAG\_FRE: Frame error flag
  - I2S\_FLAG\_CHSIDE: Channel Side flag
  - I2S\_FLAG\_BSY: Busy flag

**Return value:**

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

**\_\_HAL\_I2S\_CLEAR\_OVRFLAG****Description:**

- Clears the I2S OVR pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.

**Return value:**

- None

**\_\_HAL\_I2S\_CLEAR\_UDRFLAG****Description:**

- Clears the I2S UDR pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the I2S Handle.

**Return value:**

- None

**I2S 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

## 31 HAL IRDA Generic Driver

### 31.1 IRDA Firmware driver registers structures

#### 31.1.1 IRDA\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t Parity*
- *uint16\_t Mode*
- *uint8\_t Prescaler*
- *uint16\_t PowerMode*

##### Field Documentation

- ***uint32\_t IRDA\_InitTypeDef::BaudRate***  
This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hirda->Init.BaudRate)))
- ***uint32\_t IRDA\_InitTypeDef::WordLength***  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [IRDAEx\\_Word\\_Length](#)
- ***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).
- ***uint16\_t IRDA\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [IRDA\\_Mode](#)
- ***uint8\_t IRDA\_InitTypeDef::Prescaler***  
Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.  
**Note:**Prescaler value 0 is forbidden
- ***uint16\_t IRDA\_InitTypeDef::PowerMode***  
Specifies the IRDA power mode. This parameter can be a value of [IRDA\\_Low\\_Power](#)

#### 31.1.2 IRDA\_HandleTypeDef

##### Data Fields

- *USART\_TypeDef \* Instance*
- *IRDA\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *uint16\_t TxXferSize*

- *uint16\_t TxXferCount*
- *uint8\_t \*pRxBuffPtr*
- *uint16\_t RxXferSize*
- *uint16\_t RxXferCount*
- *uint16\_t Mask*
- *DMA\_HandleTypeDef \* hdmatx*
- *DMA\_HandleTypeDef \* hdmarx*
- *HAL\_LockTypeDef Lock*
- *\_\_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*
- *uint16\_t IRDA\_HandleTypeDef::Mask*
- *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*

## 31.2 IRDA Firmware driver API description

### 31.2.1 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

### 31.2.2 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\(\)\*](#)

### 31.2.3 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 API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_IRDA\_TxCpltCallback(), HAL\_IRDA\_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL\_IRDA\_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode API's are :
  - HAL\_IRDA\_Transmit()
  - HAL\_IRDA\_Receive()
3. Non-Blocking mode API's with Interrupt are :
  - HAL\_IRDA\_Transmit\_IT()
  - HAL\_IRDA\_Receive\_IT()
  - HAL\_IRDA\_IRQHandler()
  - IRDA\_Transmit\_IT()
  - IRDA\_Receive\_IT()
4. Non-Blocking mode functions with DMA are :
  - HAL\_IRDA\_Transmit\_DMA()

- HAL\_IRDA\_Receive\_DMA()
- 5. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
  - HAL\_IRDA\_TxCpltCallback()
  - HAL\_IRDA\_RxCpltCallback()
  - HAL\_IRDA\_ErrorCallback()

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\\_DMABase\(\)](#)
- [HAL\\_IRDA\\_DMABaseResume\(\)](#)
- [HAL\\_IRDA\\_DMABaseStop\(\)](#)
- [HAL\\_IRDA\\_IRQHandler\(\)](#)
- [HAL\\_IRDA\\_TxHalfCpltCallback\(\)](#)
- [HAL\\_IRDA\\_TxCpltCallback\(\)](#)
- [HAL\\_IRDA\\_RxHalfCpltCallback\(\)](#)
- [HAL\\_IRDA\\_RxCpltCallback\(\)](#)
- [HAL\\_IRDA\\_ErrorCallback\(\)](#)

### 31.2.4 Peripheral Control functions

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

- HAL\_IRDA\_GetState() API can be helpful to check in run-time the state of the IRDA peripheral.
- IRDA\_SetConfig() API is used to configure the IRDA communications parameters.

This section contains the following APIs:

- [HAL\\_IRDA\\_GetState\(\)](#)
- [HAL\\_IRDA\\_GetError\(\)](#)

### 31.2.5 HAL\_IRDA\_Init

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)</b>
Function Description	Initializes the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>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>

### 31.2.6 HAL\_IRDA\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)</b>
Function Description	DeInitializes the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hirda</b>: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA</li> </ul>

module.

Return values

- HAL status

### 31.2.7 HAL\_IRDA\_MspInit

Function Name **void HAL\_IRDA\_MspInit (IRDA\_HandleTypeDef \* hirda)**

Function Description IRDA MSP Init.

Parameters

- **hirda:** pointer to a IRDA\_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- None

### 31.2.8 HAL\_IRDA\_MspDeInit

Function Name **void HAL\_IRDA\_MspDeInit (IRDA\_HandleTypeDef \* hirda)**

Function Description IRDA MSP DeInit.

Parameters

- **hirda:** pointer to a IRDA\_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- None

### 31.2.9 HAL\_IRDA\_Transmit

Function Name **HAL\_StatusTypeDef HAL\_IRDA\_Transmit (IRDA\_HandleTypeDef \* hirda, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

Function Description Sends an amount of data in blocking mode.

Parameters

- **hirda:** pointer to a IRDA\_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Specify timeout value

Return values

- HAL status

### 31.2.10 HAL\_IRDA\_Receive

Function Name **HAL\_StatusTypeDef HAL\_IRDA\_Receive (IRDA\_HandleTypeDef \* hirda, uint8\_t \* pData, uint16\_t Size, uint32\_t Timeout)**

Function Description Receive an amount of data in blocking mode.

Parameters

- **hirda:** pointer to a IRDA\_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Specify timeout value

Return values

- HAL status

### 31.2.11 HAL\_IRDA\_Transmit\_IT

Function Name **HAL\_StatusTypeDef HAL\_IRDA\_Transmit\_IT**  
(IRDA\_HandleTypeDef \* hirda, uint8\_t \* pData, uint16\_t Size)

Function Description Send an amount of data in non blocking mode.

Parameters

- **hirda**: pointer to a IRDA\_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent

Return values

- HAL status

### 31.2.12 HAL\_IRDA\_Receive\_IT

Function Name **HAL\_StatusTypeDef HAL\_IRDA\_Receive\_IT**  
(IRDA\_HandleTypeDef \* hirda, uint8\_t \* pData, uint16\_t Size)

Function Description Receives an amount of data in non blocking mode.

Parameters

- **hirda**: pointer to a IRDA\_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be received

Return values

- HAL status

### 31.2.13 HAL\_IRDA\_Transmit\_DMA

Function Name **HAL\_StatusTypeDef HAL\_IRDA\_Transmit\_DMA**  
(IRDA\_HandleTypeDef \* hirda, uint8\_t \* pData, uint16\_t Size)

Function Description Sends an amount of data in non blocking mode.

Parameters

- **hirda**: pointer to a IRDA\_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be sent

Return values

- HAL status

### 31.2.14 HAL\_IRDA\_Receive\_DMA

Function Name **HAL\_StatusTypeDef HAL\_IRDA\_Receive\_DMA**  
(IRDA\_HandleTypeDef \* hirda, uint8\_t \* pData, uint16\_t Size)

Function Description Receives an amount of data in non blocking mode.

Parameters

- **hirda**: pointer to a IRDA\_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData**: Pointer to data buffer
- **Size**: Amount of data to be received



Return values	<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>

### 31.2.15 HAL\_IRDA\_DMABase

Function Name	<b>HAL_StatusTypeDef HAL_IRDA_DMABase (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>

### 31.2.16 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>

### 31.2.17 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>

### 31.2.18 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>

### 31.2.19 HAL\_IRDA\_TxHalfCpltCallback

Function Name	<b>void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</b>
---------------	--

Function Description	Tx Transfer complete callbacks.
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>• None</li></ul>

### 31.2.20 HAL\_IRDA\_TxCpltCallback

Function Name	<b>void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Tx Half Transfer completed callbacks.
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 USART module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 31.2.21 HAL\_IRDA\_RxHalfCpltCallback

Function Name	<b>void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Rx Transfer complete callbacks.
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>• None</li></ul>

### 31.2.22 HAL\_IRDA\_RxCpltCallback

Function Name	<b>void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	Rx Half Transfer complete callbacks.
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>• None</li></ul>

### 31.2.23 HAL\_IRDA\_ErrorCallback

Function Name	<b>void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)</b>
Function Description	IRDA error callbacks.
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>• None</li></ul>

### 31.2.24 HAL\_IRDA\_GetState

Function Name	<b>HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)</b>
Function Description	Returns the IRDA state.
Parameters	<ul style="list-style-type: none"> <li><b>hirda:</b> pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

### 31.2.25 HAL\_IRDA\_GetError

Function Name	<b>uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)</b>
Function Description	Return the IRDA error code.
Parameters	<ul style="list-style-type: none"> <li><b>hirda:</b> : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>IRDA Error Code</li> </ul>

## 31.3 IRDA Firmware driver defines

### 31.3.1 IRDA

#### **IRDA DMA Rx**

IRDA\_DMA\_RX\_DISABLE

IRDA\_DMA\_RX\_ENABLE

#### **IRDA DMA Tx**

IRDA\_DMA\_TX\_DISABLE

IRDA\_DMA\_TX\_ENABLE

#### **IRDA Error Code**

HAL_IRDA_ERROR_NONE	No error
HAL_IRDA_ERROR_PE	Parity error
HAL_IRDA_ERROR_NE	Noise error
HAL_IRDA_ERROR_FE	frame error
HAL_IRDA_ERROR_ORE	Overrun error
HAL_IRDA_ERROR_DMA	DMA transfer error

#### **IRDA Exported Macros**

<b>__HAL_IRDA_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset IRDA handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__:</b> specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.</li> </ul> <b>Return value:</b>
--------------------------------------	---

**\_\_HAL\_IRDA\_GET\_FLAG**

- None

**Description:**

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2. UART peripheral
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - IRDA\_FLAG\_REACK: Receive enable acknowledge flag
  - IRDA\_FLAG\_TEACK: Transmit enable acknowledge flag
  - IRDA\_FLAG\_BUSY: Busy flag
  - IRDA\_FLAG\_ABRF: Auto Baud rate detection flag
  - IRDA\_FLAG\_ABRE: Auto Baud rate detection error flag
  - IRDA\_FLAG\_TXE: Transmit data register empty flag
  - IRDA\_FLAG\_TC: Transmission Complete flag
  - IRDA\_FLAG\_RXNE: Receive data register not empty flag
  - IRDA\_FLAG\_IDLE: Idle Line detection flag
  - IRDA\_FLAG\_ORE: OverRun Error flag
  - IRDA\_FLAG\_NE: Noise Error flag
  - IRDA\_FLAG\_FE: Framing Error flag
  - IRDA\_FLAG\_PE: Parity Error flag

**Return value:**

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

**\_\_HAL\_IRDA\_ENABLE\_IT****Description:**

- Enable the specified IRDA interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2. UART peripheral
- **\_\_INTERRUPT\_\_**: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register

- not empty interrupt
- IRDA\_IT\_IDLE: Idle line detection interrupt
- IRDA\_IT\_PE: Parity Error interrupt
- IRDA\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

**Description:**

- Disable the specified IRDA interrupt.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- `__INTERRUPT__`: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register not empty interrupt
  - IRDA\_IT\_IDLE: Idle line detection interrupt
  - IRDA\_IT\_PE: Parity Error interrupt
  - IRDA\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- `__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

`__HAL_IRDA_DISABLE_IT``__HAL_IRDA_GET_IT`

- IRDA\_IT\_ORE: OverRun Error interrupt
- IRDA\_IT\_NE: Noise Error interrupt
- IRDA\_IT\_FE: Framing Error interrupt
- IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

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

**Description:**

- Check whether the specified IRDA interrupt source is enabled.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- \_\_IT\_\_: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA\_IT\_TXE: Transmit Data Register empty interrupt
  - IRDA\_IT\_TC: Transmission complete interrupt
  - IRDA\_IT\_RXNE: Receive Data register not empty interrupt
  - IRDA\_IT\_IDLE: Idle line detection interrupt
  - IRDA\_IT\_ORE: OverRun Error interrupt
  - IRDA\_IT\_NE: Noise Error interrupt
  - IRDA\_IT\_FE: Framing Error interrupt
  - IRDA\_IT\_PE: Parity Error interrupt

**Return value:**

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

**Description:**

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- \_\_IT\_CLEAR\_\_: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - IRDA\_CLEAR\_PEF: Parity Error Clear Flag
  - IRDA\_CLEAR\_FEF: Framing Error Clear Flag
  - IRDA\_CLEAR\_NEF: Noise detected Clear Flag
  - IRDA\_CLEAR\_OREF: OverRun Error

`__HAL_IRDA_GET_IT_SOURCE`

`__HAL_IRDA_CLEAR_IT`

- Clear Flag
- IRDA\_CLEAR\_TCF: Transmission Complete Clear Flag

**Return value:**

- None

**Description:**

- Set a specific IRDA request flag.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.
- `__REQ__`: specifies the request flag to set. This parameter can be one of the following values:
  - IRDA\_AUTOBAUD\_REQUEST: Auto-Baud Rate Request
  - IRDA\_RXDATA\_FLUSH\_REQUEST: Receive Data flush Request
  - IRDA\_TXDATA\_FLUSH\_REQUEST: Transmit data flush Request

**Return value:**

- None

**Description:**

- Enable UART/USART associated to IRDA Handle.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

**Description:**

- Disable UART/USART associated to IRDA Handle.

**Parameters:**

- `__HANDLE__`: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

`__HAL_IRDA_SEND_REQ`

`__HAL_IRDA_ENABLE`

`__HAL_IRDA_DISABLE`

**IRDA Flags**

`IRDA_FLAG_REACK`

IRDA\_FLAG\_TEACK

IRDA\_FLAG\_BUSY

IRDA\_FLAG\_ABRF

IRDA\_FLAG\_ABRE

IRDA\_FLAG\_TXE

IRDA\_FLAG\_TC

IRDA\_FLAG\_RXNE

IRDA\_FLAG\_ORE

IRDA\_FLAG\_NE

IRDA\_FLAG\_FE

IRDA\_FLAG\_PE

***IRDA Interruption Mask***

IRDA\_IT\_MASK

***IRDA Interrupt definition***

IRDA\_IT\_PE

IRDA\_IT\_TXE

IRDA\_IT\_TC

IRDA\_IT\_RXNE

IRDA\_IT\_IDLE

IRDA\_IT\_ERR

IRDA\_IT\_ORE

IRDA\_IT\_NE

IRDA\_IT\_FE

***IRDA IT CLEAR Flags***

IRDA\_CLEAR\_PEF      Parity Error Clear Flag

IRDA\_CLEAR\_FEF      Framing Error Clear Flag

IRDA\_CLEAR\_NEF      Noise detected Clear Flag

IRDA\_CLEAR\_OREF      OverRun Error Clear Flag

IRDA\_CLEAR\_TCF      Transmission Complete Clear Flag

***IRDA Low Power***

IRDA\_POWERMODE\_NORMAL

IRDA\_POWERMODE\_LOWPOWER

***IRDA Mode***

IRDA\_MODE\_DISABLE

IRDA\_MODE\_ENABLE

***IRDA One Bit***



IRDA\_ONE\_BIT\_SAMPLE\_DISABLE

IRDA\_ONE\_BIT\_SAMPLE\_ENABLE

**IRDA Parity**

IRDA\_PARITY\_NONE

IRDA\_PARITY\_EVEN

IRDA\_PARITY\_ODD

**IRDA Private Constants**

TEACK\_REACK\_TIMEOUT

HAL\_IRDA\_TXDMA\_TIMEOUTVALUE

IRDA\_CR1\_FIELDS

**IRDA Private Macros**

IS\_IRDA\_BAUDRATE

**Description:**

- Ensure that IRDA Baud rate is less or equal to maximum value.

**Parameters:**

- `__BAUDRATE__`: specifies the IRDA Baudrate set by the user.

**Return value:**

- True: or False

IS\_IRDA\_PRESCALER

**Description:**

- Ensure that IRDA prescaler value is strictly larger than 0.

**Parameters:**

- `__PRESCALER__`: specifies the IRDA prescaler value set by the user.

**Return value:**

- True: or False

IS\_IRDA\_PARITY

IS\_IRDA\_TX\_RX\_MODE

IS\_IRDA\_POWERMODE

IS\_IRDA\_STATE

IS\_IRDA\_MODE

IS\_IRDA\_ONE\_BIT\_SAMPLE

IS\_IRDA\_DMA\_TX

IS\_IRDA\_DMA\_RX

IS\_IRDA\_REQUEST\_PARAMETER

**IRDA Request Parameters**

---

IRDA_AUTOBAUD_REQUEST	Auto-Baud Rate Request
IRDA_RXDATA_FLUSH_REQUEST	Receive Data flush Request
IRDA_TXDATA_FLUSH_REQUEST	Transmit data flush Request

**IRDA State**

IRDA\_STATE\_DISABLE

IRDA\_STATE\_ENABLE

**IRDA Transfer Mode**

IRDA\_MODE\_RX

IRDA\_MODE\_TX

IRDA\_MODE\_TX\_RX

## 32 HAL IRDA Extension Driver

### 32.1 IRDAEx Firmware driver defines

#### 32.1.1 IRDAEx

##### *IRDAEx Private Macros*

##### IRDA\_GETCLOCKSOURCE

##### Description:

- Reports the IRDA clock source.

##### Parameters:

- `__HANDLE__`: specifies the IRDA Handle
- `__CLOCKSOURCE__`: output variable

##### Return value:

- IRDA: clocking source, written in `__CLOCKSOURCE__`.

##### IRDA\_MASK\_COMPUTATION

##### Description:

- Reports the mask to apply to retrieve the received data according to the word length and to the parity bits activation.

##### Parameters:

- `__HANDLE__`: specifies the IRDA Handle

##### Return value:

- mask: to apply to USART RDR register value.

##### IS\_IRDA\_WORD\_LENGTH

##### *IRDAEx Word Length*

##### IRDA\_WORDLENGTH\_7B

##### IRDA\_WORDLENGTH\_8B

##### IRDA\_WORDLENGTH\_9B

## 33 HAL IWDG Generic Driver

### 33.1 IWDG Firmware driver registers structures

#### 33.1.1 IWDG\_InitTypeDef

##### Data Fields

- *uint32\_t Prescaler*
- *uint32\_t Reload*
- *uint32\_t Window*

##### Field Documentation

- ***uint32\_t IWDG\_InitTypeDef::Prescaler***  
Select the prescaler of the IWDG. This parameter can be a value of [IWDG\\_Prescaler](#)
- ***uint32\_t IWDG\_InitTypeDef::Reload***  
Specifies the IWDG down-counter reload value. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x0FFF
- ***uint32\_t IWDG\_InitTypeDef::Window***  
Specifies the window value to be compared to the down-counter. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x0FFF

#### 33.1.2 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

## 33.2 IWDG Firmware driver API description

### 33.2.1 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
- Manage Window option
- Initialize the IWDG MSP
- DeInitialize IWDG MSP

This section contains the following APIs:

- [HAL\\_IWDG\\_Init\(\)](#)
- [HAL\\_IWDG\\_MsplInit\(\)](#)

### 33.2.2 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\(\)](#)

### 33.2.3 Peripheral State functions

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

This section contains the following APIs:

- [HAL\\_IWDG\\_GetState\(\)](#)

### 33.2.4 HAL\_IWDG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)</b>
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>

### 33.2.5 HAL\_IWDG\_MsplInit

Function Name	<b>void HAL_IWDG_MsplInit (IWDG_HandleTypeDef * hiwdg)</b>
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>

**33.2.6 HAL\_IWDG\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)</b>
Function Description	Starts the IWDG.
Parameters	<ul style="list-style-type: none"> <li><b>hiwdg</b>: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**33.2.7 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>

**33.2.8 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>

**33.3 IWDG Firmware driver defines****33.3.1 IWDG*****IWDG Exported Macros***

<b>__HAL_IWDG_RESET_HANDLE_STATE</b>	<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><b>__HANDLE__</b>: IWDG handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
<b>__HAL_IWDG_START</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Enables the IWDG peripheral.</li> </ul> <b>Parameters:</b>

**\_\_HAL\_IWDG\_RELOAD\_COUNTER**

- **\_\_HANDLE\_\_**: IWDG handle

**Return value:**

- None

**Description:**

- Reloads IWDG counter with value defined in the reload register (write access to IWDG\_PR and IWDG\_RLR registers disabled).

**Parameters:**

- **\_\_HANDLE\_\_**: IWDG handle

**Return value:**

- None

**\_\_HAL\_IWDG\_GET\_FLAG****Description:**

- Gets the selected IWDG's flag status.

**Parameters:**

- **\_\_HANDLE\_\_**: IWDG handle
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - IWDG\_FLAG\_PVU: Watchdog counter reload value update flag
  - IWDG\_FLAG\_RVU: Watchdog counter prescaler value flag
  - IWDG\_FLAG\_WVU: Watchdog counter window value flag

**Return value:**

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

**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 Defines**

HAL\_IWDG\_DEFAULT\_TIMEOUT

IWDG\_KEY\_RELOAD

IWDG Reload Counter Enable

IWDG\_KEY\_ENABLE

IWDG Peripheral Enable

IWDG_KEY_WRITE_ACCESS_ENABLE	IWDG KR Write Access Enable
IWDG_KEY_WRITE_ACCESS_DISABLE	IWDG KR Write Access Disable
IWDG_FLAG_PVU	Watchdog counter prescaler value update flag
IWDG_FLAG_RVU	Watchdog counter reload value update flag
IWDG_FLAG_WVU	Watchdog counter window value update flag

**IWDG Private Macros**

IWDG_ENABLE_WRITE_ACCESS	<b>Description:</b> <ul style="list-style-type: none"> <li>Enables write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: IWDG handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
IWDG_DISABLE_WRITE_ACCESS	<b>Description:</b> <ul style="list-style-type: none"> <li>Disables write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: IWDG handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
IS_IWDG_PRESCALER	<b>Description:</b> <ul style="list-style-type: none"> <li>Check IWDG prescaler value.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__PRESCALER__: IWDG prescaler value</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
IS_IWDG_RELOAD	<b>Description:</b> <ul style="list-style-type: none"> <li>Check IWDG reload value.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__RELOAD__: IWDG reload value</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
IS_IWDG_WINDOW	<b>Description:</b> <ul style="list-style-type: none"> <li>Check IWDG window value.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__WINDOW__: IWDG window value</li> </ul>



**Return value:**

- None

***IWDG Window***

IWDG\_WINDOW\_DISABLE

## 34 HAL LPTIM Generic Driver

### 34.1 LPTIM Firmware driver registers structures

#### 34.1.1 LPTIM\_ClockConfigTypeDef

##### Data Fields

- *uint32\_t Source*
- *uint32\_t Prescaler*

##### Field Documentation

- *uint32\_t LPTIM\_ClockConfigTypeDef::Source*  
Selects the clock source. This parameter can be a value of [LPTIM\\_Clock\\_Source](#)
- *uint32\_t LPTIM\_ClockConfigTypeDef::Prescaler*  
Specifies the counter clock Prescaler. This parameter can be a value of [LPTIM\\_Clock\\_Prescaler](#)

#### 34.1.2 LPTIM\_ULPClockConfigTypeDef

##### Data Fields

- *uint32\_t Polarity*
- *uint32\_t SampleTime*

##### Field Documentation

- *uint32\_t LPTIM\_ULPClockConfigTypeDef::Polarity*  
Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of [LPTIM\\_Clock\\_Polarity](#)
- *uint32\_t LPTIM\_ULPClockConfigTypeDef::SampleTime*  
Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of [LPTIM\\_Clock\\_Sample\\_Time](#)

#### 34.1.3 LPTIM\_TriggerConfigTypeDef

##### Data Fields

- *uint32\_t Source*
- *uint32\_t ActiveEdge*
- *uint32\_t SampleTime*

**Field Documentation**

- ***uint32\_t LPTIM\_TriggerConfigTypeDef::Source***  
Selects the Trigger source. This parameter can be a value of [LPTIM\\_Trigger\\_Source](#)
- ***uint32\_t LPTIM\_TriggerConfigTypeDef::ActiveEdge***  
Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [LPTIM\\_External\\_Trigger\\_Polarity](#)
- ***uint32\_t LPTIM\_TriggerConfigTypeDef::SampleTime***  
Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of [LPTIM\\_Trigger\\_Sample\\_Time](#)

**34.1.4 LPTIM\_InitTypeDef****Data Fields**

- ***LPTIM\_ClockConfigTypeDef Clock***
- ***LPTIM\_ULPClockConfigTypeDef UltraLowPowerClock***
- ***LPTIM\_TriggerConfigTypeDef Trigger***
- ***uint32\_t OutputPolarity***
- ***uint32\_t UpdateMode***
- ***uint32\_t CounterSource***

**Field Documentation**

- ***LPTIM\_ClockConfigTypeDef LPTIM\_InitTypeDef::Clock***  
Specifies the clock parameters
- ***LPTIM\_ULPClockConfigTypeDef LPTIM\_InitTypeDef::UltraLowPowerClock***  
Specifies the Ultra Low Power clock parameters
- ***LPTIM\_TriggerConfigTypeDef LPTIM\_InitTypeDef::Trigger***  
Specifies the Trigger parameters
- ***uint32\_t LPTIM\_InitTypeDef::OutputPolarity***  
Specifies the Output polarity. This parameter can be a value of [LPTIM\\_Output\\_Polarity](#)
- ***uint32\_t LPTIM\_InitTypeDef::UpdateMode***  
Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of [LPTIM\\_Updating\\_Mode](#)
- ***uint32\_t LPTIM\_InitTypeDef::CounterSource***  
Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of [LPTIM\\_Counter\\_Source](#)

**34.1.5 LPTIM\_HandleTypeDef****Data Fields**

- ***LPTIM\_TypeDef \* Instance***
- ***LPTIM\_InitTypeDef Init***

- ***HAL\_StatusTypeDef Status***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_LPTIM\_StateTypeDef State***

#### Field Documentation

- ***LPTIM\_TypeDef\* LPTIM\_HandleTypeDef::Instance***  
Register base address
- ***LPTIM\_InitTypeDef LPTIM\_HandleTypeDef::Init***  
LPTIM required parameters
- ***HAL\_StatusTypeDef LPTIM\_HandleTypeDef::Status***  
LPTIM peripheral status
- ***HAL\_LockTypeDef LPTIM\_HandleTypeDef::Lock***  
LPTIM locking object
- ***\_\_IO HAL\_LPTIM\_StateTypeDef LPTIM\_HandleTypeDef::State***  
LPTIM peripheral state

## 34.2 LPTIM Firmware driver API description

### 34.2.1 How to use this driver

The LPTIM HAL driver can be used as follows:

1. Initialize the LPTIM low level resources by implementing the HAL\_LPTIM\_MspInit():
  - a. Enable the LPTIM interface clock using \_\_LPTIMx\_CLK\_ENABLE().
  - b. In case of using interrupts (e.g. HAL\_LPTIM\_PWM\_Start\_IT()): (+) Configure the LPTIM interrupt priority using HAL\_NVIC\_SetPriority(). (+) Enable the LPTIM IRQ handler using HAL\_NVIC\_EnableIRQ(). (+) In LPTIM IRQ handler, call HAL\_LPTIM\_IRQHandler().
2. Initialize the LPTIM HAL using HAL\_LPTIM\_Init(). This function configures mainly:
  - a. The instance: LPTIM1.
  - b. Clock: the counter clock. - Source : it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE, LSI or MSI). - Prescaler: select the clock divider.
  - c. UltraLowPowerClock : To be used only if the ULPTIM is selected as counter clock source. - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected. - SampleTime: clock sampling time to configure the clock glitch filter.
  - d. Trigger: How the counter start. - Source: trigger can be software or one of the hardware triggers. - ActiveEdge : only for hardware trigger. - SampleTime : trigger sampling time to configure the trigger glitch filter.
  - e. OutputPolarity : 2 opposite polarities are possibles.
  - f. UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period.
3. Six modes are available:
  - a. PWM Mode: To generate a PWM signal with specified period and pulse, call HAL\_LPTIM\_PWM\_Start() or HAL\_LPTIM\_PWM\_Start\_IT() for interruption mode.
  - b. One Pulse Mode: To generate pulse with specified width in response to a stimulus, call HAL\_LPTIM\_OnePulse\_Start() or HAL\_LPTIM\_OnePulse\_Start\_IT() for interruption mode.
  - c. Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare

- match occurs. To start this mode, call HAL\_LPTIM\_SetOnce\_Start() or HAL\_LPTIM\_SetOnce\_Start\_IT() for interruption mode.
- d. Encoder Mode: To use the encoder interface call HAL\_LPTIM\_Encoder\_Start() or HAL\_LPTIM\_Encoder\_Start\_IT() for interruption mode.
  - e. Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call HAL\_LPTIM\_TimeOut\_Start\_IT() or HAL\_LPTIM\_TimeOut\_Start\_IT() for interruption mode.
  - f. Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL\_LPTIM\_Counter\_Start() or HAL\_LPTIM\_Counter\_Start\_IT() for interruption mode.
4. User can stop any process by calling the corresponding API: HAL\_LPTIM\_Xxx\_Stop() or HAL\_LPTIM\_Xxx\_Stop\_IT() if the process is already started in interruption mode.
  5. Call HAL\_LPTIM\_DeInit() to deinitialize the LPTIM peripheral.

The LPTIM HAL driver can be used as follows: (#)Initialize the LPTIM low level resources by implementing the HAL\_LPTIM\_MspInit(): (##) Enable the LPTIM interface clock using \_\_LPTIMx\_CLK\_ENABLE(). (##) In case of using interrupts (e.g. HAL\_LPTIM\_PWM\_Start\_IT()):

- Configure the LPTIM interrupt priority using HAL\_NVIC\_SetPriority().
- Enable the LPTIM IRQ handler using HAL\_NVIC\_EnableIRQ().
- In LPTIM IRQ handler, call HAL\_LPTIM\_IRQHandler(). (#)Initialize the LPTIM HAL using HAL\_LPTIM\_Init(). This function configures mainly:
  - a. The instance: LPTIM1.
  - b. Clock: the counter clock. - Source : it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE, LSI or MSI). - Prescaler: select the clock divider.
  - c. UltraLowPowerClock : To be used only if the ULPTIM is selected as counter clock source. - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected. - SampleTime: clock sampling time to configure the clock glitch filter.
  - d. Trigger: How the counter start. - Source: trigger can be software or one of the hardware triggers. - ActiveEdge : only for hardware trigger. - SampleTime : trigger sampling time to configure the trigger glitch filter.
  - e. OutputPolarity : 2 opposite polarities are possibles.
  - f. UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. (#)Six modes are available:
    - g. PWM Mode: To generate a PWM signal with specified period and pulse, call HAL\_LPTIM\_PWM\_Start() or HAL\_LPTIM\_PWM\_Start\_IT() for interruption mode.
    - h. One Pulse Mode: To generate pulse with specified width in response to a stimulus, call HAL\_LPTIM\_OnePulse\_Start() or HAL\_LPTIM\_OnePulse\_Start\_IT() for interruption mode.
    - i. Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare match occurs. To start this mode, call HAL\_LPTIM\_SetOnce\_Start() or HAL\_LPTIM\_SetOnce\_Start\_IT() for interruption mode.
    - j. Encoder Mode: To use the encoder interface call HAL\_LPTIM\_Encoder\_Start() or HAL\_LPTIM\_Encoder\_Start\_IT() for interruption mode.
    - k. Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call

HAL\_LPTIM\_TimeOut\_Start\_IT() or HAL\_LPTIM\_TimeOut\_Start\_IT() for interruption mode.

- I. Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL\_LPTIM\_Counter\_Start() or HAL\_LPTIM\_Counter\_Start\_IT() for interruption mode. (#) User can stop any process by calling the corresponding API: HAL\_LPTIM\_Xxx\_Stop() or HAL\_LPTIM\_Xxx\_Stop\_IT() if the process is already started in interruption mode. (#)Call HAL\_LPTIM\_DeInit() to deinitialize the LPTIM peripheral.

### 34.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM\_InitTypeDef and creates the associated handle.
- DeInitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- DeInitialize LPTIM MSP.

This section contains the following APIs:

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

### 34.2.3 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- [\*HAL\\_LPTIM\\_PWM\\_Start\(\)\*](#)
- [\*HAL\\_LPTIM\\_PWM\\_Stop\(\)\*](#)
- [\*HAL\\_LPTIM\\_PWM\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_LPTIM\\_PWM\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_LPTIM\\_OnePulse\\_Start\(\)\*](#)
- [\*HAL\\_LPTIM\\_OnePulse\\_Stop\(\)\*](#)
- [\*HAL\\_LPTIM\\_OnePulse\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_LPTIM\\_OnePulse\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_LPTIM\\_SetOnce\\_Start\(\)\*](#)
- [\*HAL\\_LPTIM\\_SetOnce\\_Stop\(\)\*](#)
- [\*HAL\\_LPTIM\\_SetOnce\\_Start\\_IT\(\)\*](#)

- [HAL\\_LPTIM\\_SetOnce\\_Stop\\_IT\(\)](#)
- [HAL\\_LPTIM\\_Encoder\\_Start\(\)](#)
- [HAL\\_LPTIM\\_Encoder\\_Stop\(\)](#)
- [HAL\\_LPTIM\\_Encoder\\_Start\\_IT\(\)](#)
- [HAL\\_LPTIM\\_Encoder\\_Stop\\_IT\(\)](#)
- [HAL\\_LPTIM\\_TimeOut\\_Start\(\)](#)
- [HAL\\_LPTIM\\_TimeOut\\_Stop\(\)](#)
- [HAL\\_LPTIM\\_TimeOut\\_Start\\_IT\(\)](#)
- [HAL\\_LPTIM\\_TimeOut\\_Stop\\_IT\(\)](#)
- [HAL\\_LPTIM\\_Counter\\_Start\(\)](#)
- [HAL\\_LPTIM\\_Counter\\_Stop\(\)](#)
- [HAL\\_LPTIM\\_Counter\\_Start\\_IT\(\)](#)
- [HAL\\_LPTIM\\_Counter\\_Stop\\_IT\(\)](#)

### 34.2.4 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare)value.

This section contains the following APIs:

- [HAL\\_LPTIM\\_ReadCounter\(\)](#)
- [HAL\\_LPTIM\\_ReadAutoReload\(\)](#)
- [HAL\\_LPTIM\\_ReadCompare\(\)](#)

### 34.2.5 LPTIM IRQ handler

This section provides LPTIM IRQ handler function.

This section contains the following APIs:

- [HAL\\_LPTIM\\_IRQHandler\(\)](#)
- [HAL\\_LPTIM\\_CompareMatchCallback\(\)](#)
- [HAL\\_LPTIM\\_AutoReloadMatchCallback\(\)](#)
- [HAL\\_LPTIM\\_TriggerCallback\(\)](#)
- [HAL\\_LPTIM\\_CompareWriteCallback\(\)](#)
- [HAL\\_LPTIM\\_AutoReloadWriteCallback\(\)](#)
- [HAL\\_LPTIM\\_DirectionUpCallback\(\)](#)
- [HAL\\_LPTIM\\_DirectionDownCallback\(\)](#)

### 34.2.6 Peripheral State functions

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

This section contains the following APIs:

- [HAL\\_LPTIM\\_GetState\(\)](#)

### 34.2.7 HAL\_LPTIM\_Init

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef *hlptim)</b>
Function Description	Initializes the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.

- |               |                               |
|---------------|-------------------------------|
| Parameters    | • <b>hlptim:</b> LPTIM handle |
| Return values | • HAL status                  |

### 34.2.8 HAL\_LPTIM\_DeInit

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)</b> |
| Function Description | DeInitializes the LPTIM peripheral.                                      |
| Parameters           | • <b>hlptim:</b> LPTIM handle  |
| Return values        | • HAL status   |

### 34.2.9 HAL\_LPTIM\_MspInit

- |                      |  |
|----------------------|--|
| Function Name        | <b>void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)</b> |
| Function Description | Initializes the LPTIM MSP.                                   |
| Parameters           | • <b>hlptim:</b> LPTIM handle                                |
| Return values        | • None   |

### 34.2.10 HAL\_LPTIM\_MspDeInit

- |                      |  |
|----------------------|--|
| Function Name        | <b>void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)</b> |
| Function Description | DeInitializes LPTIM MSP.                                       |
| Parameters           | • <b>hlptim:</b> LPTIM handle                                  |
| Return values        | • None   |

### 34.2.11 HAL\_LPTIM\_PWM\_Start

- |                      |   |
|----------------------|---|
| Function Name        | <b>HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)</b>  |
| Function Description | Starts the LPTIM PWM generation.  |
| Parameters           | • <b>hlptim:</b> : LPTIM handle<br>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.<br>• <b>Pulse:</b> : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF. |
| Return values        | • HAL status  |

### 34.2.12 HAL\_LPTIM\_PWM\_Stop

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim)</b> |
| Function Description | Stops the LPTIM PWM generation.  |
| Parameters           | • <b>hlptim:</b> : LPTIM handle  |
| Return values        | • HAL status   |



**34.2.13 HAL\_LPTIM\_PWM\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT</b> (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM PWM generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF</li> <li>• <b>Pulse:</b> : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**34.2.14 HAL\_LPTIM\_PWM\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT</b> (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM PWM generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**34.2.15 HAL\_LPTIM\_OnePulse\_Start**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start</b> (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM One pulse generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li>• <b>Pulse:</b> : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**34.2.16 HAL\_LPTIM\_OnePulse\_Stop**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop</b> (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM One pulse generation.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**34.2.17 HAL\_LPTIM\_OnePulse\_Start\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT</b> (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
---------------	---

Function Description	Starts the LPTIM One pulse generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li>• <b>Pulse:</b> : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 34.2.18 HAL\_LPTIM\_OnePulse\_Stop\_IT

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT</b> (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM One pulse generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 34.2.19 HAL\_LPTIM\_SetOnce\_Start

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start</b> (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM in Set once mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li>• <b>Pulse:</b> : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 34.2.20 HAL\_LPTIM\_SetOnce\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop</b> (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM Set once mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 34.2.21 HAL\_LPTIM\_SetOnce\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT</b> (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM Set once mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> <li>• <b>Pulse:</b> : Specifies the compare value. This parameter must</li> </ul>

be a value between 0x0000 and 0xFFFF.

Return values

- HAL status

### 34.2.22 HAL\_LPTIM\_SetOnce\_Stop\_IT

Function Name **HAL\_StatusTypeDef HAL\_LPTIM\_SetOnce\_Stop\_IT (LPTIM\_HandleTypeDef \* hlptim)**

Function Description Stops the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** : LPTIM handle

Return values

- HAL status

### 34.2.23 HAL\_LPTIM\_Encoder\_Start

Function Name **HAL\_StatusTypeDef HAL\_LPTIM\_Encoder\_Start (LPTIM\_HandleTypeDef \* hlptim, uint32\_t Period)**

Function Description Starts the Encoder interface.

Parameters

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- HAL status

### 34.2.24 HAL\_LPTIM\_Encoder\_Stop

Function Name **HAL\_StatusTypeDef HAL\_LPTIM\_Encoder\_Stop (LPTIM\_HandleTypeDef \* hlptim)**

Function Description Stops the Encoder interface.

Parameters

- **hlptim:** : LPTIM handle

Return values

- HAL status

### 34.2.25 HAL\_LPTIM\_Encoder\_Start\_IT

Function Name **HAL\_StatusTypeDef HAL\_LPTIM\_Encoder\_Start\_IT (LPTIM\_HandleTypeDef \* hlptim, uint32\_t Period)**

Function Description Starts the Encoder interface in interrupt mode.

Parameters

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- HAL status

### 34.2.26 HAL\_LPTIM\_Encoder\_Stop\_IT

Function Name **HAL\_StatusTypeDef HAL\_LPTIM\_Encoder\_Stop\_IT (LPTIM\_HandleTypeDef \* hlptim)**

Function Description Stops the Encoder interface in interrupt mode.

Parameters

- **hlptim:** : LPTIM handle

Return values

- HAL status

### 34.2.27 HAL\_LPTIM\_TimeOut\_Start

**Function Name** HAL\_StatusTypeDef HAL\_LPTIM\_TimeOut\_Start (LPTIM\_HandleTypeDef \* hlptim, uint32\_t Period, uint32\_t Timeout)

**Function Description** Starts the Timeout function.

**Parameters**

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Timeout:** : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- HAL status

### 34.2.28 HAL\_LPTIM\_TimeOut\_Stop

**Function Name** HAL\_StatusTypeDef HAL\_LPTIM\_TimeOut\_Stop (LPTIM\_HandleTypeDef \* hlptim)

**Function Description** Stops the Timeout function.

**Parameters**

- **hlptim:** : LPTIM handle

Return values

- HAL status

### 34.2.29 HAL\_LPTIM\_TimeOut\_Start\_IT

**Function Name** HAL\_StatusTypeDef HAL\_LPTIM\_TimeOut\_Start\_IT (LPTIM\_HandleTypeDef \* hlptim, uint32\_t Period, uint32\_t Timeout)

**Function Description** Starts the Timeout function in interrupt mode.

**Parameters**

- **hlptim:** : LPTIM handle
- **Period:** : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
- **Timeout:** : Specifies the TimeOut value to rest the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- HAL status

### 34.2.30 HAL\_LPTIM\_TimeOut\_Stop\_IT

**Function Name** HAL\_StatusTypeDef HAL\_LPTIM\_TimeOut\_Stop\_IT (LPTIM\_HandleTypeDef \* hlptim)

**Function Description** Stops the Timeout function in interrupt mode.

**Parameters**

- **hlptim:** : LPTIM handle

Return values

- HAL status

### 34.2.31 HAL\_LPTIM\_Counter\_Start

Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Counter_Start</b> (LPTIM_HandleTypeDef * hlptim, uint32_t Period)
Function Description	Starts the Counter mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>34.2.32 HAL_LPTIM_Counter_Stop</b>	
Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Counter_Stop</b> (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the Counter mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>34.2.33 HAL_LPTIM_Counter_Start_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT</b> (LPTIM_HandleTypeDef * hlptim, uint32_t Period)
Function Description	Starts the Counter mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> <li>• <b>Period:</b> : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>34.2.34 HAL_LPTIM_Counter_Stop_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT</b> (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the Counter mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>34.2.35 HAL_LPTIM_ReadCounter</b>	
Function Name	<b>uint32_t HAL_LPTIM_ReadCounter</b> (LPTIM_HandleTypeDef * hlptim)
Function Description	This function returns the current counter value.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Counter value.</li> </ul>
<b>34.2.36 HAL_LPTIM_ReadAutoReload</b>	
Function Name	<b>uint32_t HAL_LPTIM_ReadAutoReload</b> (LPTIM_HandleTypeDef * hlptim)

---

Function Description	This function return the current Autoreload (Period) value.
Parameters	<ul style="list-style-type: none"><li>• <b>hlptim</b>: LPTIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• Autoreload value.</li></ul>

### 34.2.37 HAL\_LPTIM\_ReadCompare

Function Name	<b>uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	This function return the current Compare (Pulse) value.
Parameters	<ul style="list-style-type: none"><li>• <b>hlptim</b>: LPTIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• Compare value.</li></ul>

### 34.2.38 HAL\_LPTIM\_IRQHandler

Function Name	<b>void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	This function handles LPTIM interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>hlptim</b>: LPTIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 34.2.39 HAL\_LPTIM\_CompareMatchCallback

Function Name	<b>void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Compare match callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hlptim</b>: : LPTIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 34.2.40 HAL\_LPTIM\_AutoReloadMatchCallback

Function Name	<b>void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Autoreload match callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hlptim</b>: : LPTIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 34.2.41 HAL\_LPTIM\_TriggerCallback

Function Name	<b>void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Trigger detected callback in non blocking mode.
Parameters	<ul style="list-style-type: none"><li>• <b>hlptim</b>: : LPTIM handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 34.2.42 HAL\_LPTIM\_CompareWriteCallback

Function Name	<b>void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Compare write callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 34.2.43 HAL\_LPTIM\_AutoReloadWriteCallback

Function Name	<b>void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Autoreload write callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 34.2.44 HAL\_LPTIM\_DirectionUpCallback

Function Name	<b>void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Direction counter changed from Down to Up callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 34.2.45 HAL\_LPTIM\_DirectionDownCallback

Function Name	<b>void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Direction counter changed from Up to Down callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> : LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 34.2.46 HAL\_LPTIM\_GetState

Function Name	<b>HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hlptim)</b>
Function Description	Returns the LPTIM state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hlptim:</b> LPTIM handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

## 34.3 LPTIM Firmware driver defines

### 34.3.1 LPTIM

*LPTIM Clock Polarity*

LPTIM\_CLOCKPOLARITY\_RISING  
LPTIM\_CLOCKPOLARITY\_FALLING  
LPTIM\_CLOCKPOLARITY\_RISING\_FALLING

**LPTIM Clock Prescaler**

LPTIM\_PRESCALER\_DIV1  
LPTIM\_PRESCALER\_DIV2  
LPTIM\_PRESCALER\_DIV4  
LPTIM\_PRESCALER\_DIV8  
LPTIM\_PRESCALER\_DIV16  
LPTIM\_PRESCALER\_DIV32  
LPTIM\_PRESCALER\_DIV64  
LPTIM\_PRESCALER\_DIV128

**LPTIM Clock Sample Time**

LPTIM\_CLOCKSAMPLETIME\_DIRECTTRANSITION  
LPTIM\_CLOCKSAMPLETIME\_2TRANSITIONS  
LPTIM\_CLOCKSAMPLETIME\_4TRANSITIONS  
LPTIM\_CLOCKSAMPLETIME\_8TRANSITIONS

**LPTIM Clock Source**

LPTIM\_CLOCKSOURCE\_APBCLK\_LPOSC  
LPTIM\_CLOCKSOURCE\_ULPTIM

**LPTIM Counter Source**

LPTIM\_COUNTERSOURCE\_INTERNAL  
LPTIM\_COUNTERSOURCE\_EXTERNAL

**LPTIM Exported Macros**

\_\_HAL\_LPTIM\_RESET\_HANDLE\_STATE

**Description:**

- Reset LPTIM handle state.

**Parameters:**

- \_\_HANDLE\_\_: LPTIM handle

**Return value:**

- None

\_\_HAL\_LPTIM\_ENABLE

**Description:**

- Enable/Disable the LPTIM peripheral.

**Parameters:**

- \_\_HANDLE\_\_: LPTIM handle

**Return value:**

- None



---

`__HAL_LPTIM_DISABLE``__HAL_LPTIM_START_CONTINUOUS`**Description:**

- Starts the LPTIM peripheral in Continuous or in single mode.

**Parameters:**

- `__HANDLE__`: DMA handle

**Return value:**

- None

`__HAL_LPTIM_START_SINGLE``__HAL_LPTIM_AUTORELOAD_SET`**Description:**

- Writes the passed parameter in the Autoreload register.

**Parameters:**

- `__HANDLE__`: LPTIM handle
- `__VALUE__`: Autoreload value

**Return value:**

- None

`__HAL_LPTIM_COMPARE_SET`**Description:**

- Writes the passed parameter in the Compare register.

**Parameters:**

- `__HANDLE__`: LPTIM handle
- `__VALUE__`: Compare value

**Return value:**

- None

`__HAL_LPTIM_GET_FLAG`**Description:**

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

**Parameters:**

- `__HANDLE__`: LPTIM handle
- `__FLAG__`: LPTIM flag to check This parameter can be a value of:
  - `LPTIM_FLAG_DOWN`: Counter direction change up Flag.
  - `LPTIM_FLAG_UP`: Counter direction change down to up Flag.
  - `LPTIM_FLAG_ARROK`: Autoreload register update OK Flag.
  - `LPTIM_FLAG_CMPOK`: Compare register update OK Flag.
  - `LPTIM_FLAG_EXTTRIG`: External trigger edge event Flag.

- LPTIM\_FLAG\_ARRM : Autoreload match Flag.
- LPTIM\_FLAG\_CMPM : Compare match Flag.

**Return value:**

- The: state of the specified flag (SET or RESET).

**Description:**

- Clears the specified LPTIM flag.

**Parameters:**

- \_\_HANDLE\_\_ : LPTIM handle.
- \_\_FLAG\_\_ : LPTIM flag to clear. This parameter can be a value of:
  - LPTIM\_FLAG\_DOWN : Counter direction change up Flag.
  - LPTIM\_FLAG\_UP : Counter direction change down to up Flag.
  - LPTIM\_FLAG\_ARROK : Autoreload register update OK Flag.
  - LPTIM\_FLAG\_CMPOK : Compare register update OK Flag.
  - LPTIM\_FLAG\_EXTTRIG : External trigger edge event Flag.
  - LPTIM\_FLAG\_ARRM : Autoreload match Flag.
  - LPTIM\_FLAG\_CMPM : Compare match Flag.

**Return value:**

- None.

**Description:**

- Enable the specified LPTIM interrupt.

**Parameters:**

- \_\_HANDLE\_\_ : LPTIM handle.
- \_\_INTERRUPT\_\_ : LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM\_IT\_DOWN : Counter direction change up Interrupt.
  - LPTIM\_IT\_UP : Counter direction change down to up Interrupt.
  - LPTIM\_IT\_ARROK : Autoreload register update OK Interrupt.
  - LPTIM\_IT\_CMPOK : Compare register update OK Interrupt.
  - LPTIM\_IT\_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM\_IT\_ARRM : Autoreload match Interrupt.
  - LPTIM\_IT\_CMPM : Compare match

\_\_HAL\_LPTIM\_CLEAR\_FLAG

\_\_HAL\_LPTIM\_ENABLE\_IT

Interrupt.

**Return value:**

- None.

**Description:**

- Disable the specified LPTIM interrupt.

**Parameters:**

- `__HANDLE__`: : LPTIM handle.
- `__INTERRUPT__`: : LPTIM interrupt to set. This parameter can be a value of:
  - `LPTIM_IT_DOWN`: Counter direction change up Interrupt.
  - `LPTIM_IT_UP`: Counter direction change down to up Interrupt.
  - `LPTIM_IT_ARROK`: Autoreload register update OK Interrupt.
  - `LPTIM_IT_CMPOK`: Compare register update OK Interrupt.
  - `LPTIM_IT_EXTTRIG`: External trigger edge event Interrupt.
  - `LPTIM_IT_ARRM`: Autoreload match Interrupt.
  - `LPTIM_IT_CMPM`: Compare match Interrupt.

**Return value:**

- None.

**Description:**

- Checks whether the specified LPTIM interrupt is set or not.

**Parameters:**

- `__HANDLE__`: : LPTIM handle.
- `__INTERRUPT__`: : LPTIM interrupt to check. This parameter can be a value of:
  - `LPTIM_IT_DOWN`: Counter direction change up Interrupt.
  - `LPTIM_IT_UP`: Counter direction change down to up Interrupt.
  - `LPTIM_IT_ARROK`: Autoreload register update OK Interrupt.
  - `LPTIM_IT_CMPOK`: Compare register update OK Interrupt.
  - `LPTIM_IT_EXTTRIG`: External trigger edge event Interrupt.
  - `LPTIM_IT_ARRM`: Autoreload match Interrupt.
  - `LPTIM_IT_CMPM`: Compare match Interrupt.

**Return value:**

`__HAL_LPTIM_DISABLE_IT`

`__HAL_LPTIM_GET_IT_SOURCE`

- Interrupt: status.

***LPTIM External Trigger Polarity***

LPTIM\_ACTIVEEDGE\_RISING

LPTIM\_ACTIVEEDGE\_FALLING

LPTIM\_ACTIVEEDGE\_RISING\_FALLING

***LPTIM Flag Definition***

LPTIM\_FLAG\_DOWN

LPTIM\_FLAG\_UP

LPTIM\_FLAG\_ARROK

LPTIM\_FLAG\_CMPOK

LPTIM\_FLAG\_EXTTRIG

LPTIM\_FLAG\_ARRM

LPTIM\_FLAG\_CMPM

***LPTIM Interrupts Definition***

LPTIM\_IT\_DOWN

LPTIM\_IT\_UP

LPTIM\_IT\_ARROK

LPTIM\_IT\_CMPOK

LPTIM\_IT\_EXTTRIG

LPTIM\_IT\_ARRM

LPTIM\_IT\_CMPM

***LPTIM Output Polarity***

LPTIM\_OUTPUTPOLARITY\_HIGH

LPTIM\_OUTPUTPOLARITY\_LOW

***LPTIM Private Macros***

IS\_LPTIM\_CLOCK\_SOURCE

IS\_LPTIM\_CLOCK\_PRESCALER

IS\_LPTIM\_CLOCK\_PRESCALERDIV1

IS\_LPTIM\_OUTPUT\_POLARITY

IS\_LPTIM\_CLOCK\_SAMPLE\_TIME

IS\_LPTIM\_CLOCK\_POLARITY

IS\_LPTIM\_TRG\_SOURCE

IS\_LPTIM\_EXT\_TRG\_POLARITY

IS\_LPTIM\_TRIG\_SAMPLE\_TIME

IS\_LPTIM\_UPDATE\_MODE

IS\_LPTIM\_COUNTER\_SOURCE

IS\_LPTIM\_AUTORELOAD

IS\_LPTIM\_COMPARE

IS\_LPTIM\_PERIOD

IS\_LPTIM\_PULSE

***LPTIM Trigger Sample Time***

LPTIM\_TRIGSAMPLETIME\_DIRECTTRANSITION

LPTIM\_TRIGSAMPLETIME\_2TRANSITIONS

LPTIM\_TRIGSAMPLETIME\_4TRANSITIONS

LPTIM\_TRIGSAMPLETIME\_8TRANSITIONS

***LPTIM Trigger Source***

LPTIM\_TRIGSOURCE\_SOFTWARE

LPTIM\_TRIGSOURCE\_0

LPTIM\_TRIGSOURCE\_1

LPTIM\_TRIGSOURCE\_2

LPTIM\_TRIGSOURCE\_3

LPTIM\_TRIGSOURCE\_4

LPTIM\_TRIGSOURCE\_5

***LPTIM Updating Mode***

LPTIM\_UPDATE\_IMMEDIATE

LPTIM\_UPDATE\_ENDOFPERIOD

## 35 HAL LTDC Generic Driver

### 35.1 LTDC Firmware driver registers structures

#### 35.1.1 LTDC\_ColorTypeDef

##### Data Fields

- *uint8\_t Blue*
- *uint8\_t Green*
- *uint8\_t Red*
- *uint8\_t Reserved*

##### Field Documentation

- *uint8\_t LTDC\_ColorTypeDef::Blue*  
Configures the blue value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- *uint8\_t LTDC\_ColorTypeDef::Green*  
Configures the green value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- *uint8\_t LTDC\_ColorTypeDef::Red*  
Configures the red value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- *uint8\_t LTDC\_ColorTypeDef::Reserved*  
Reserved 0xFF

#### 35.1.2 LTDC\_InitTypeDef

##### Data Fields

- *uint32\_t HSPolarity*
- *uint32\_t VSPolarity*
- *uint32\_t DEPolarity*
- *uint32\_t PCPolarity*
- *uint32\_t HorizontalSync*
- *uint32\_t VerticalSync*
- *uint32\_t AccumulatedHBP*
- *uint32\_t AccumulatedVBP*
- *uint32\_t AccumulatedActiveW*
- *uint32\_t AccumulatedActiveH*
- *uint32\_t TotalWidth*
- *uint32\_t TotalHeigh*
- *LTDC\_ColorTypeDef Backcolor*

##### Field Documentation

- ***uint32\_t LTDC\_InitTypeDef::HSPolarity***  
configures the horizontal synchronization polarity. This parameter can be one value of ***LTDC\_HS\_POLARITY***
- ***uint32\_t LTDC\_InitTypeDef::VSPolarity***  
configures the vertical synchronization polarity. This parameter can be one value of ***LTDC\_VS\_POLARITY***
- ***uint32\_t LTDC\_InitTypeDef::DEPolarity***  
configures the data enable polarity. This parameter can be one of value of ***LTDC\_DE\_POLARITY***
- ***uint32\_t LTDC\_InitTypeDef::PCPolarity***  
configures the pixel clock polarity. This parameter can be one of value of ***LTDC\_PC\_POLARITY***
- ***uint32\_t LTDC\_InitTypeDef::HorizontalSync***  
configures the number of Horizontal synchronization width. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFF.
- ***uint32\_t LTDC\_InitTypeDef::VerticalSync***  
configures the number of Vertical synchronization height. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0x7FF.
- ***uint32\_t LTDC\_InitTypeDef::AccumulatedHBP***  
configures the accumulated horizontal back porch width. This parameter must be a number between Min\_Data = LTDC\_HorizontalSync and Max\_Data = 0xFFF.
- ***uint32\_t LTDC\_InitTypeDef::AccumulatedVBP***  
configures the accumulated vertical back porch height. This parameter must be a number between Min\_Data = LTDC\_VerticalSync and Max\_Data = 0x7FF.
- ***uint32\_t LTDC\_InitTypeDef::AccumulatedActiveW***  
configures the accumulated active width. This parameter must be a number between Min\_Data = LTDC\_AccumulatedHBP and Max\_Data = 0xFFF.
- ***uint32\_t LTDC\_InitTypeDef::AccumulatedActiveH***  
configures the accumulated active height. This parameter must be a number between Min\_Data = LTDC\_AccumulatedVBP and Max\_Data = 0x7FF.
- ***uint32\_t LTDC\_InitTypeDef::TotalWidth***  
configures the total width. This parameter must be a number between Min\_Data = LTDC\_AccumulatedActiveW and Max\_Data = 0xFFF.
- ***uint32\_t LTDC\_InitTypeDef::TotalHeigh***  
configures the total height. This parameter must be a number between Min\_Data = LTDC\_AccumulatedActiveH and Max\_Data = 0x7FF.
- ***LTDC\_ColorTypeDef LTDC\_InitTypeDef::Backcolor***  
Configures the background color.

### 35.1.3 LTDC\_LayerCfgTypeDef

#### Data Fields

- ***uint32\_t WindowX0***
- ***uint32\_t WindowX1***
- ***uint32\_t WindowY0***
- ***uint32\_t WindowY1***
- ***uint32\_t PixelFormat***
- ***uint32\_t Alpha***
- ***uint32\_t Alpha0***
- ***uint32\_t BlendingFactor1***
- ***uint32\_t BlendingFactor2***

- ***uint32\_t FBStartAddress***
- ***uint32\_t ImageWidth***
- ***uint32\_t ImageHeight***
- ***LTDC\_ColorTypeDef Backcolor***

#### Field Documentation

- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowX0***  
Configures the Window Horizontal Start Position. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowX1***  
Configures the Window Horizontal Stop Position. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowY0***  
Configures the Window vertical Start Position. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0xFFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::WindowY1***  
Configures the Window vertical Stop Position. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::PixelFormat***  
Specifies the pixel format. This parameter can be one of value of [LTDC\\_Pixelformat](#)
- ***uint32\_t LTDC\_LayerCfgTypeDef::Alpha***  
Specifies the constant alpha used for blending. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::Alpha0***  
Configures the default alpha value. This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0xFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::BlendingFactor1***  
Select the blending factor 1. This parameter can be one of value of [LTDC\\_BlendingFactor1](#)
- ***uint32\_t LTDC\_LayerCfgTypeDef::BlendingFactor2***  
Select the blending factor 2. This parameter can be one of value of [LTDC\\_BlendingFactor2](#)
- ***uint32\_t LTDC\_LayerCfgTypeDef::FBStartAddress***  
Configures the color frame buffer address
- ***uint32\_t LTDC\_LayerCfgTypeDef::ImageWidth***  
Configures the color frame buffer line length. This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0x1FFF.
- ***uint32\_t LTDC\_LayerCfgTypeDef::ImageHeight***  
Specifies the number of line in frame buffer. This parameter must be a number between Min\_Data = 0x000 and Max\_Data = 0x7FF.
- ***LTDC\_ColorTypeDef LTDC\_LayerCfgTypeDef::Backcolor***  
Configures the layer background color.

### 35.1.4 LTDC\_HandleTypeDef

#### Data Fields

- ***LTDC\_TypeDef \* Instance***
- ***LTDC\_InitTypeDef Init***
- ***LTDC\_LayerCfgTypeDef LayerCfg***
- ***HAL\_LockTypeDef Lock***



- `__IO HAL_LTDC_StateTypeDef State`
- `__IO uint32_t ErrorCode`

#### Field Documentation

- **`LTDC_TypeDef* LTDC_HandleTypeDef::Instance`**  
LTDC Register base address
- **`LTDC_InitTypeDef LTDC_HandleTypeDef::Init`**  
LTDC parameters
- **`LTDC_LayerCfgTypeDef LTDC_HandleTypeDef::LayerCfg[MAX_LAYER]`**  
LTDC Layers parameters
- **`HAL_LockTypeDef LTDC_HandleTypeDef::Lock`**  
LTDC Lock
- **`__IO HAL_LTDC_StateTypeDef LTDC_HandleTypeDef::State`**  
LTDC state
- **`__IO uint32_t LTDC_HandleTypeDef::ErrorCode`**  
LTDC Error code

## 35.2 LTDC Firmware driver API description

### 35.2.1 How to use this driver

1. Program the required configuration through the following parameters: the LTDC timing, the horizontal and vertical polarity, the pixel clock polarity, Data Enable polarity and the LTDC background color value using `HAL_LTDC_Init()` function
2. Program the required configuration through the following parameters: the pixel format, the blending factors, input alpha value, the window size and the image size using `HAL_LTDC_ConfigLayer()` function for foreground or/and background layer.
3. Optionally, configure and enable the CLUT using `HAL_LTDC_ConfigCLUT()` and `HAL_LTDC_EnableCLUT` functions.
4. Optionally, enable the Dither using `HAL_LTDC_EnableDither()`.
5. Optionally, configure and enable the Color keying using `HAL_LTDC_ConfigColorKeying()` and `HAL_LTDC_EnableColorKeying` functions.
6. Optionally, configure LineInterrupt using `HAL_LTDC_ProgramLineEvent()` function
7. If needed, reconfigure and change the pixel format value, the alpha value value, the window size, the window position and the layer start address for foreground or/and background layer using respectively the following functions:  
`HAL_LTDC_SetPixelFormat()`, `HAL_LTDC_SetAlpha()`,  
`HAL_LTDC_SetWindowSize()`, `HAL_LTDC_SetWindowPosition()`,  
`HAL_LTDC_SetAddress`.
8. To control LTDC state you can use the following function: `HAL_LTDC_GetState()`

#### LTDC HAL driver macros list

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

- `__HAL_LTDC_ENABLE`: Enable the LTDC.
- `__HAL_LTDC_DISABLE`: Disable the LTDC.
- `__HAL_LTDC_LAYER_ENABLE`: Enable the LTDC Layer.
- `__HAL_LTDC_LAYER_DISABLE`: Disable the LTDC Layer.
- `__HAL_LTDC_RELOAD_CONFIG`: Reload Layer Configuration.
- `__HAL_LTDC_GET_FLAG`: Get the LTDC pending flags.

- `__HAL_LTDC_CLEAR_FLAG`: Clear the LTDC pending flags.
- `__HAL_LTDC_ENABLE_IT`: Enable the specified LTDC interrupts.
- `__HAL_LTDC_DISABLE_IT`: Disable the specified LTDC interrupts.
- `__HAL_LTDC_GET_IT_SOURCE`: Check whether the specified LTDC interrupt has occurred or not.



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

### 35.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC
- De-initialize the LTDC

This section contains the following APIs:

- [\*HAL\\_LTDC\\_Init\(\)\*](#)
- [\*HAL\\_LTDC\\_DeInit\(\)\*](#)
- [\*HAL\\_LTDC\\_MspInit\(\)\*](#)
- [\*HAL\\_LTDC\\_MspDeInit\(\)\*](#)
- [\*HAL\\_LTDC\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_LTDC\\_LineEvenCallback\(\)\*](#)

### 35.2.3 IO operation functions

This section provides function allowing to:

- Handle LTDC interrupt request

This section contains the following APIs:

- [\*HAL\\_LTDC\\_IRQHandler\(\)\*](#)
- [\*HAL\\_LTDC\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_LTDC\\_LineEvenCallback\(\)\*](#)

### 35.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the LTDC foreground or/and background parameters.
- Set the active layer.
- Configure the color keying.
- Configure the C-LUT.
- Enable / Disable the color keying.
- Enable / Disable the C-LUT.
- Update the layer position.
- Update the layer size.
- Update pixel format on the fly.
- Update transparency on the fly.
- Update address on the fly.

This section contains the following APIs:

- [\*HAL\\_LTDC\\_ConfigLayer\(\)\*](#)
- [\*HAL\\_LTDC\\_ConfigColorKeying\(\)\*](#)

- [HAL\\_LTDC\\_ConfigCLUT\(\)](#)
- [HAL\\_LTDC\\_EnableColorKeying\(\)](#)
- [HAL\\_LTDC\\_DisableColorKeying\(\)](#)
- [HAL\\_LTDC\\_EnableCLUT\(\)](#)
- [HAL\\_LTDC\\_DisableCLUT\(\)](#)
- [HAL\\_LTDC\\_EnableDither\(\)](#)
- [HAL\\_LTDC\\_DisableDither\(\)](#)
- [HAL\\_LTDC\\_SetWindowSize\(\)](#)
- [HAL\\_LTDC\\_SetWindowPosition\(\)](#)
- [HAL\\_LTDC\\_SetPixelFormat\(\)](#)
- [HAL\\_LTDC\\_SetAlpha\(\)](#)
- [HAL\\_LTDC\\_SetAddress\(\)](#)
- [HAL\\_LTDC\\_ProgramLineEvent\(\)](#)

### 35.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the LTDC state.
- Get error code.

This section contains the following APIs:

- [HAL\\_LTDC\\_GetState\(\)](#)
- [HAL\\_LTDC\\_GetError\(\)](#)

### 35.2.6 HAL\_LTDC\_Init

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

### 35.2.7 HAL\_LTDC\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_DeInit (LTDC_HandleTypeDef * hltdc)</b>
Function Description	Deinitializes the LTDC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 35.2.8 HAL\_LTDC\_MspInit

Function Name	<b>void HAL_LTDC_MspInit (LTDC_HandleTypeDef * hltdc)</b>
Function Description	Initializes the LTDC MSP.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>

Return values

- None

### 35.2.9 HAL\_LTDC\_MspDeInit

Function Name **void HAL\_LTDC\_MspDeInit (LTDC\_HandleTypeDef \* hltdc)**

Function Description DeInitializes the LTDC MSP.

Parameters

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

Return values

- None

### 35.2.10 HAL\_LTDC\_ErrorCallback

Function Name **void HAL\_LTDC\_ErrorCallback (LTDC\_HandleTypeDef \* hltdc)**

Function Description Error LTDC callback.

Parameters

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

Return values

- None

### 35.2.11 HAL\_LTDC\_LineEvenCallback

Function Name **void HAL\_LTDC\_LineEvenCallback (LTDC\_HandleTypeDef \* hltdc)**

Function Description Line Event callback.

Parameters

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

Return values

- None

### 35.2.12 HAL\_LTDC\_IRQHandler

Function Name **void HAL\_LTDC\_IRQHandler (LTDC\_HandleTypeDef \* hltdc)**

Function Description Handles LTDC interrupt request.

Parameters

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

Return values

- HAL status

### 35.2.13 HAL\_LTDC\_ErrorCallback

Function Name **void HAL\_LTDC\_ErrorCallback (LTDC\_HandleTypeDef \* hltdc)**

Function Description Error LTDC callback.

Parameters

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

Return values

- None

### 35.2.14 HAL\_LTDC\_LineEvenCallback

Function Name **void HAL\_LTDC\_LineEvenCallback (LTDC\_HandleTypeDef \* hltdc)**

**hltdc)**

Function Description	Line Event callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**35.2.15 HAL\_LTDC\_ConfigLayer**

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_ConfigLayer</b> (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)
Function Description	Configure the LTDC Layer according to the specified parameters in the LTDC_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>pLayerCfg</b>: pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.</li> <li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**35.2.16 HAL\_LTDC\_ConfigColorKeying**

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying</b> (LTDC_HandleTypeDef * hltdc, uint32_t RGBValue, uint32_t LayerIdx)
Function Description	Configure the color keying.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>RGBValue</b>: the color key value</li> <li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**35.2.17 HAL\_LTDC\_ConfigCLUT**

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_ConfigCLUT</b> (LTDC_HandleTypeDef * hltdc, uint32_t * pCLUT, uint32_t CLUTSize, uint32_t LayerIdx)
Function Description	Load the color lookup table.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>pCLUT</b>: pointer to the color lookup table address.</li> <li>• <b>CLUTSize</b>: the color lookup table size.</li> <li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 35.2.18 HAL\_LTDC\_EnableColorKeying

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_EnableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)</b>
Function Description	Enable the color keying.
Parameters	<ul style="list-style-type: none"><li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li><li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 35.2.19 HAL\_LTDC\_DisableColorKeying

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_DisableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)</b>
Function Description	Disable the color keying.
Parameters	<ul style="list-style-type: none"><li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li><li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 35.2.20 HAL\_LTDC\_EnableCLUT

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_EnableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)</b>
Function Description	Enable the color lookup table.
Parameters	<ul style="list-style-type: none"><li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li><li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 35.2.21 HAL\_LTDC\_DisableCLUT

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_DisableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)</b>
Function Description	Disable the color lookup table.
Parameters	<ul style="list-style-type: none"><li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li><li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

### 35.2.22 HAL\_LTDC\_EnableDither

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_EnableDither (LTDC_HandleTypeDef * hltdc)</b>
---------------	---

Function Description	Enables Dither.
Parameters	<ul style="list-style-type: none"> <li><b>hltdc:</b> pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 35.2.23 HAL\_LTDC\_DisableDither

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_DisableDither</b> (LTDC_HandleTypeDef * hltdc)
Function Description	Disables Dither.
Parameters	<ul style="list-style-type: none"> <li><b>hltdc:</b> pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 35.2.24 HAL\_LTDC\_SetWindowSize

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_SetWindowSize</b> (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)
Function Description	Set the LTDC window size.
Parameters	<ul style="list-style-type: none"> <li><b>hltdc:</b> pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li><b>XSize:</b> LTDC Pixel per line</li> <li><b>YSize:</b> LTDC Line number</li> <li><b>LayerIdx:</b> LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 35.2.25 HAL\_LTDC\_SetWindowPosition

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_SetWindowPosition</b> (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)
Function Description	Set the LTDC window position.
Parameters	<ul style="list-style-type: none"> <li><b>hltdc:</b> pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li><b>X0:</b> LTDC window X offset</li> <li><b>Y0:</b> LTDC window Y offset</li> <li><b>LayerIdx:</b> LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 35.2.26 HAL\_LTDC\_SetPixelFormat

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_SetPixelFormat</b> (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)
---------------	---

Function Description	Reconfigure the pixel format.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>Pixelformat</b>: new pixel format value.</li> <li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 35.2.27 HAL\_LTDC\_SetAlpha

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_SetAlpha</b> (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)
Function Description	Reconfigure the layer alpha value.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>Alpha</b>: new alpha value.</li> <li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 35.2.28 HAL\_LTDC\_SetAddress

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_SetAddress</b> (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)
Function Description	Reconfigure the frame buffer Address.
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>Address</b>: new address value.</li> <li>• <b>LayerIdx</b>: LTDC Layer index. This parameter can be one of the following values: 0 or 1.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 35.2.29 HAL\_LTDC\_ProgramLineEvent

Function Name	<b>HAL_StatusTypeDef HAL_LTDC_ProgramLineEvent</b> (LTDC_HandleTypeDef * hltdc, uint32_t Line)
Function Description	Define the position of the line interrupt .
Parameters	<ul style="list-style-type: none"> <li>• <b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> <li>• <b>Line</b>: Line Interrupt Position.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 35.2.30 HAL\_LTDC\_GetState

Function Name	<b>HAL_LTDC_StateTypeDef HAL_LTDC_GetState</b> (LTDC_HandleTypeDef * hltdc)
---------------	--



Function Description	Return the LTDC state.
Parameters	<ul style="list-style-type: none"> <li><b>hltdc</b>: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

### 35.2.31 HAL\_LTDC\_GetError

Function Name	<b>uint32_t HAL_LTDC_GetError (LTDC_HandleTypeDef * hltdc)</b>
Function Description	Return the LTDC error code.
Parameters	<ul style="list-style-type: none"> <li><b>hltdc</b>: : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>LTDC Error Code</li> </ul>

## 35.3 LTDC Firmware driver defines

### 35.3.1 LTDC

#### *LTDC Alpha*

LTDC\_ALPHA LTDC Cte Alpha mask

#### *LTDC BACK COLOR*

LTDC\_COLOR Color mask

#### *LTDC Blending Factor1*

LTDC\_BLENDING\_FACTOR1\_CA Blending factor : Cte Alpha

LTDC\_BLENDING\_FACTOR1\_PAxCA Blending factor : Cte Alpha x Pixel Alpha

#### *LTDC Blending Factor2*

LTDC\_BLENDING\_FACTOR2\_CA Blending factor : Cte Alpha

LTDC\_BLENDING\_FACTOR2\_PAxCA Blending factor : Cte Alpha x Pixel Alpha

#### *LTDC DE POLARITY*

LTDC\_DEPOLARITY\_AL Data Enable, is active low.

LTDC\_DEPOLARITY\_AH Data Enable, is active high.

#### *LTDC Error Code*

HAL\_LTDC\_ERROR\_NONE LTDC No error

HAL\_LTDC\_ERROR\_TE LTDC Transfer error

HAL\_LTDC\_ERROR\_FU LTDC FIFO Underrun

HAL\_LTDC\_ERROR\_TIMEOUT LTDC Timeout error

#### *LTDC Exported Macros*

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

- Reset LTDC handle state.

**Parameters:**

- \_\_HANDLE\_\_**: specifies the LTDC

`__HAL_LTDC_ENABLE`

handle.

**Return value:**

- None

**Description:**

- Enable the LTDC.

**Parameters:**

- `__HANDLE__`: LTDC handle

**Return value:**

- None.

`__HAL_LTDC_DISABLE`

**Description:**

- Disable the LTDC.

**Parameters:**

- `__HANDLE__`: LTDC handle

**Return value:**

- None.

`__HAL_LTDC_LAYER_ENABLE`

**Description:**

- Enable the LTDC Layer.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__LAYER__`: Specify the layer to be enabled This parameter can be 0 or 1

**Return value:**

- None.

`__HAL_LTDC_LAYER_DISABLE`

**Description:**

- Disable the LTDC Layer.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__LAYER__`: Specify the layer to be disabled This parameter can be 0 or 1

**Return value:**

- None.

`__HAL_LTDC_RELOAD_CONFIG`

**Description:**

- Reload Layer Configuration.

**Parameters:**

- `__HANDLE__`: LTDC handle

**Return value:**

- None.

---

**\_\_HAL\_LTDC\_GET\_FLAG****Description:**

- Get the LTDC pending flags.

**Parameters:**

- **\_\_HANDLE\_\_**: LTDC handle
- **\_\_FLAG\_\_**: Get the specified flag. This parameter can be any combination of the following values:
  - LTDC\_FLAG\_LI: Line Interrupt flag
  - LTDC\_FLAG\_FU: FIFO Underrun Interrupt flag
  - LTDC\_FLAG\_TE: Transfer Error interrupt flag
  - LTDC\_FLAG\_RR: Register Reload Interrupt Flag

**Return value:**

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

**\_\_HAL\_LTDC\_CLEAR\_FLAG****Description:**

- Clears the LTDC pending flags.

**Parameters:**

- **\_\_HANDLE\_\_**: LTDC handle
- **\_\_FLAG\_\_**: specifies the flag to clear. This parameter can be any combination of the following values:
  - LTDC\_FLAG\_LI: Line Interrupt flag
  - LTDC\_FLAG\_FU: FIFO Underrun Interrupt flag
  - LTDC\_FLAG\_TE: Transfer Error interrupt flag
  - LTDC\_FLAG\_RR: Register Reload Interrupt Flag

**Return value:**

- None

**\_\_HAL\_LTDC\_ENABLE\_IT****Description:**

- Enables the specified LTDC interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: LTDC handle
- **\_\_INTERRUPT\_\_**: specifies the LTDC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - LTDC\_IT\_LI: Line Interrupt flag
  - LTDC\_IT\_FU: FIFO Underrun Interrupt flag
  - LTDC\_IT\_TE: Transfer Error interrupt flag
  - LTDC\_IT\_RR: Register Reload

## Interrupt Flag

`__HAL_LTDC_DISABLE_IT`**Return value:**

- None

**Description:**

- Disables the specified LTDC interrupts.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__INTERRUPT__`: specifies the LTDC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - LTDC\_IT\_LI: Line Interrupt flag
  - LTDC\_IT\_FU: FIFO Underrun Interrupt flag
  - LTDC\_IT\_TE: Transfer Error interrupt flag
  - LTDC\_IT\_RR: Register Reload Interrupt Flag

**Return value:**

- None

`__HAL_LTDC_GET_IT_SOURCE`**Description:**

- Checks whether the specified LTDC interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: LTDC handle
- `__INTERRUPT__`: specifies the LTDC interrupt source to check. This parameter can be one of the following values:
  - LTDC\_IT\_LI: Line Interrupt flag
  - LTDC\_IT\_FU: FIFO Underrun Interrupt flag
  - LTDC\_IT\_TE: Transfer Error interrupt flag
  - LTDC\_IT\_RR: Register Reload Interrupt Flag

**Return value:**

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

**LTDC Exported Types**`MAX_LAYER`**LTDC Flag**`LTDC_FLAG_LI``LTDC_FLAG_FU``LTDC_FLAG_TE`

LTDC\_FLAG\_RR

### **LTDC HS POLARITY**

LTDC\_HSPOLARITY\_AL Horizontal Synchronization is active low.

LTDC\_HSPOLARITY\_AH Horizontal Synchronization is active high.

### **LTDC Interrupts**

LTDC\_IT\_LI

LTDC\_IT\_FU

LTDC\_IT\_TE

LTDC\_IT\_RR

### **LTDC LAYER Config**

LTDC\_STOPPOSITION LTDC Layer stop position

LTDC\_STARTPOSITION LTDC Layer start position

LTDC\_COLOR\_FRAME\_BUFFER LTDC Layer Line length

LTDC\_LINE\_NUMBER LTDC Layer Line number

### **LTDC PC POLARITY**

LTDC\_PCPOLARITY\_IPC input pixel clock.

LTDC\_PCPOLARITY\_IIPC inverted input pixel clock.

### **LTDC Pixel format**

LTDC\_PIXEL\_FORMAT\_ARGB8888 ARGB8888 LTDC pixel format

LTDC\_PIXEL\_FORMAT\_RGB888 RGB888 LTDC pixel format

LTDC\_PIXEL\_FORMAT\_RGB565 RGB565 LTDC pixel format

LTDC\_PIXEL\_FORMAT\_ARGB1555 ARGB1555 LTDC pixel format

LTDC\_PIXEL\_FORMAT\_ARGB4444 ARGB4444 LTDC pixel format

LTDC\_PIXEL\_FORMAT\_L8 L8 LTDC pixel format

LTDC\_PIXEL\_FORMAT\_AL44 AL44 LTDC pixel format

LTDC\_PIXEL\_FORMAT\_AL88 AL88 LTDC pixel format

### **LTDC Private Macros**

LTDC\_LAYER

IS\_LTDC\_LAYER

IS\_LTDC\_HSPOL

IS\_LTDC\_VSPOL

IS\_LTDC\_DEPOL

IS\_LTDC\_PCPOL

IS\_LTDC\_HSYNC

IS\_LTDC\_VSYNC

IS\_LTDC\_AHBP

IS\_LTDC\_AVBP  
IS\_LTDC\_AAW  
IS\_LTDC\_AAH  
IS\_LTDC\_TOTALW  
IS\_LTDC\_TOTALH  
IS\_LTDC\_BLUEVALUE  
IS\_LTDC\_GREENVALUE  
IS\_LTDC\_REDVALUE  
IS\_LTDC\_BLENDING\_FACTOR1  
IS\_LTDC\_BLENDING\_FACTOR2  
IS\_LTDC\_PIXEL\_FORMAT  
IS\_LTDC\_ALPHA  
IS\_LTDC\_HCONFIGST  
IS\_LTDC\_HCONFIGSP  
IS\_LTDC\_VCONFIGST  
IS\_LTDC\_VCONFIGSP  
IS\_LTDC\_CFBP  
IS\_LTDC\_CFBLL  
IS\_LTDC\_CFBLNBR  
IS\_LTDC\_LIPOS

**LTDC SYNC**

LTDC\_HORIZONTALSYNC    Horizontal synchronization width.

LTDC\_VERTICALSYNC      Vertical synchronization height.

**LTDC VS POLARITY**

LTDC\_VSPOLARITY\_AL    Vertical Synchronization is active low.

LTDC\_VSPOLARITY\_AH    Vertical Synchronization is active high.

## 36 HAL NAND Generic Driver

### 36.1 NAND Firmware driver registers structures

#### 36.1.1 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*

#### 36.1.2 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

#### 36.1.3 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

### 36.1.4 NAND\_HandleTypeDef

#### Data Fields

- ***FMC\_NAND\_TypeDef \* Instance***
- ***FMC\_NAND\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_NAND\_StateTypeDef State***
- ***NAND\_InfoTypeDef Info***

#### Field Documentation

- ***FMC\_NAND\_TypeDef\* NAND\_HandleTypeDef::Instance***  
Register base address
- ***FMC\_NAND\_InitTypeDef NAND\_HandleTypeDef::Init***  
NAND device control configuration parameters
- ***HAL\_LockTypeDef NAND\_HandleTypeDef::Lock***  
NAND locking object
- ***\_\_IO HAL\_NAND\_StateTypeDef NAND\_HandleTypeDef::State***  
NAND device access state
- ***NAND\_InfoTypeDef NAND\_HandleTypeDef::Info***  
NAND characteristic information structure

## 36.2 NAND Firmware driver API description

### 36.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC/FSMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function `HAL_NAND_Init()` with control and timing parameters for both common and attribute spaces.



- Read NAND flash memory maker and device IDs using the function `HAL_NAND_Read_ID()`. The read information is stored in the `NAND_ID_TypeDef` structure declared by the function caller.
- Access NAND flash memory by read/write operations using the functions `HAL_NAND_Read_Page()/HAL_NAND_Read_SpareArea()`, `HAL_NAND_Write_Page()/HAL_NAND_Write_SpareArea()` to read/write page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the `HAL_NAND_Info_TypeDef` structure. The read/write address information is contained by the `Nand_Address_Typedef` structure passed as parameter.
- Perform NAND flash Reset chip operation using the function `HAL_NAND_Reset()`.
- Perform NAND flash erase block operation using the function `HAL_NAND_Erase_Block()`. The erase block address information is contained in the `Nand_Address_Typedef` structure passed as parameter.
- Read the NAND flash status operation using the function `HAL_NAND_Read_Status()`.
- You can also control the NAND device by calling the control APIs `HAL_NAND_ECC_Enable()/ HAL_NAND_ECC_Disable()` to respectively enable/disable the ECC code correction feature or the function `HAL_NAND_GetECC()` to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function `HAL_NAND_GetState()`



This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

### 36.2.2 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\\_DeInit\(\)\*](#)
- [\*HAL\\_NAND\\_MspInit\(\)\*](#)
- [\*HAL\\_NAND\\_MspDeInit\(\)\*](#)
- [\*HAL\\_NAND\\_IRQHandler\(\)\*](#)
- [\*HAL\\_NAND\\_ITCallback\(\)\*](#)

### 36.2.3 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\(\)\*](#)

### 36.2.4 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\(\)](#)

### 36.2.5 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\(\)](#)

### 36.2.6 HAL\_NAND\_Init

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)</b>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.7 HAL\_NAND\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)</b>
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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 36.2.8 HAL\_NAND\_MspInit

Function Name	<b>void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)</b>
Function Description	NAND MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnand</b>: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**36.2.9 HAL\_NAND\_MspDeInit**

Function Name	<b>void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnd)</b>
Function Description	NAND MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li><b>hnd:</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>

**36.2.10 HAL\_NAND\_IRQHandler**

Function Name	<b>void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnd)</b>
Function Description	This function handles NAND device interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hnd:</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>

**36.2.11 HAL\_NAND\_ITCallback**

Function Name	<b>void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnd)</b>
Function Description	NAND interrupt feature callback.
Parameters	<ul style="list-style-type: none"> <li><b>hnd:</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>

**36.2.12 HAL\_NAND\_Read\_ID**

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnd, NAND_IDTypeDef * pNAND_ID)</b>
Function Description	Read the NAND memory electronic signature.
Parameters	<ul style="list-style-type: none"> <li><b>hnd:</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>

**36.2.13 HAL\_NAND\_Reset**

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnd)</b>
Function Description	NAND memory reset.
Parameters	<ul style="list-style-type: none"> <li><b>hnd:</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>

**36.2.14 HAL\_NAND\_Read\_Page**

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Read_Page</b> (NAND_HandleTypeDef * hnd, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)
Function Description	Read Page(s) from NAND memory block.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnd</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>

### 36.2.15 HAL\_NAND\_Write\_Page

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Write_Page</b> (NAND_HandleTypeDef * hnd, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToWrite)
Function Description	Write Page(s) to NAND memory block.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnd</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>

### 36.2.16 HAL\_NAND\_Read\_SpareArea

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Read_SpareArea</b> (NAND_HandleTypeDef * hnd, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)
Function Description	Read Spare area(s) from NAND memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnd</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>

### 36.2.17 HAL\_NAND\_Write\_SpareArea

Function Name	<b>HAL_StatusTypeDef HAL_NAND_Write_SpareArea</b> (NAND_HandleTypeDef * hnd, NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)
Function Description	Write Spare area(s) to NAND memory.
Parameters	<ul style="list-style-type: none"> <li>• <b>hnd</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

### 36.2.18 HAL\_NAND\_Erase\_Block

Function Name **HAL\_StatusTypeDef HAL\_NAND\_Erase\_Block (NAND\_HandleTypeDef \* hnd, NAND\_AddressTypeDef \* pAddress)**

Function Description NAND memory Block erase.

Parameters

- **hnd**: pointer to a NAND\_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress**: : pointer to NAND address structure

Return values

- HAL status

### 36.2.19 HAL\_NAND\_Read\_Status

Function Name **uint32\_t HAL\_NAND\_Read\_Status (NAND\_HandleTypeDef \* hnd)**

Function Description NAND memory read status.

Parameters

- **hnd**: pointer to a NAND\_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- NAND status

### 36.2.20 HAL\_NAND\_Address\_Inc

Function Name **uint32\_t HAL\_NAND\_Address\_Inc (NAND\_HandleTypeDef \* hnd, NAND\_AddressTypeDef \* pAddress)**

Function Description Increment the NAND memory address.

Parameters

- **hnd**: 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

### 36.2.21 HAL\_NAND\_ECC\_Enable

Function Name **HAL\_StatusTypeDef HAL\_NAND\_ECC\_Enable (NAND\_HandleTypeDef \* hnd)**

Function Description Enables dynamically NAND ECC feature.

Parameters

- **hnd**: pointer to a NAND\_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- HAL status

### 36.2.22 HAL\_NAND\_ECC\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnd)</b>
Function Description	Disables dynamically FMC_NAND ECC feature.
Parameters	<ul style="list-style-type: none"> <li><b>hnd</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>

### 36.2.23 HAL\_NAND\_GetECC

Function Name	<b>HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnd, uint32_t * ECCval, uint32_t Timeout)</b>
Function Description	Disables dynamically NAND ECC feature.
Parameters	<ul style="list-style-type: none"> <li><b>hnd</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>

### 36.2.24 HAL\_NAND\_GetState

Function Name	<b>HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hnd)</b>
Function Description	return the NAND state
Parameters	<ul style="list-style-type: none"> <li><b>hnd</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>

### 36.2.25 HAL\_NAND\_Read\_Status

Function Name	<b>uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hnd)</b>
Function Description	NAND memory read status.
Parameters	<ul style="list-style-type: none"> <li><b>hnd</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>

## 36.3 NAND Firmware driver defines

### 36.3.1 NAND

#### *NAND Exported Macros*

<b>__HAL_NAND_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset NAND handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__</b>: specifies the NAND</li> </ul>
--------------------------------------	---

handle.

**Return value:**

- None

***NAND Private Constants***

NAND\_DEVICE  
NAND\_WRITE\_TIMEOUT  
CMD\_AREA  
ADDR\_AREA  
NAND\_CMD\_AREA\_A  
NAND\_CMD\_AREA\_B  
NAND\_CMD\_AREA\_C  
NAND\_CMD\_AREA\_TRUE1  
NAND\_CMD\_WRITE0  
NAND\_CMD\_WRITE\_TRUE1  
NAND\_CMD\_ERASE0  
NAND\_CMD\_ERASE1  
NAND\_CMD\_READID  
NAND\_CMD\_STATUS  
NAND\_CMD\_LOCK\_STATUS  
NAND\_CMD\_RESET  
NAND\_VALID\_ADDRESS  
NAND\_INVALID\_ADDRESS  
NAND\_TIMEOUT\_ERROR  
NAND\_BUSY  
NAND\_ERROR  
NAND\_READY

***NAND Private Macros***

ARRAY\_ADDRESS

**Description:**

- NAND memory address computation.

**Parameters:**

- \_\_ADDRESS\_\_: NAND memory address.
- \_\_HANDLE\_\_: : NAND handle.

**Return value:**

- NAND: Raw address value

ADDR\_1ST\_CYCLE

**Description:**

- NAND memory address cycling.

**Parameters:**

- `__ADDRESS__`: NAND memory address.

**Return value:**

- NAND: address cycling value.

`ADDR_2ND_CYCLE``ADDR_3RD_CYCLE``ADDR_4TH_CYCLE`



## 37 HAL NOR Generic Driver

### 37.1 NOR Firmware driver registers structures

#### 37.1.1 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

#### 37.1.2 NOR\_CFIDTypeDef

##### Data Fields

- *uint16\_t CFI\_1*
- *uint16\_t CFI\_2*
- *uint16\_t CFI\_3*
- *uint16\_t CFI\_4*

##### Field Documentation

- *uint16\_t NOR\_CFIDTypeDef::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\_CFIDTypeDef::CFI\_2*
- *uint16\_t NOR\_CFIDTypeDef::CFI\_3*
- *uint16\_t NOR\_CFIDTypeDef::CFI\_4*

#### 37.1.3 NOR\_HandleTypeDef

**Data Fields**

- ***FMC\_NORSRAM\_TypeDef \* Instance***
- ***FMC\_NORSRAM\_EXTENDED\_TypeDef \* Extended***
- ***FMC\_NORSRAM\_InitTypeDef Init***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_NOR\_StateTypeDef State***

**Field Documentation**

- ***FMC\_NORSRAM\_TypeDef\* NOR\_HandleTypeDef::Instance***  
Register base address
- ***FMC\_NORSRAM\_EXTENDED\_TypeDef\* NOR\_HandleTypeDef::Extended***  
Extended mode register base address
- ***FMC\_NORSRAM\_InitTypeDef NOR\_HandleTypeDef::Init***  
NOR device control configuration parameters
- ***HAL\_LockTypeDef NOR\_HandleTypeDef::Lock***  
NOR locking object
- ***\_\_IO HAL\_NOR\_StateTypeDef NOR\_HandleTypeDef::State***  
NOR device access state

## 37.2 NOR Firmware driver API description

### 37.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function `HAL_NOR_Init()` with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function `HAL_NOR_Read_ID()`. The read information is stored in the `NOR_ID_TypeDef` structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions `HAL_NOR_Read()`, `HAL_NOR_Program()`.
- Perform NOR flash erase block/chip operations using the functions `HAL_NOR_Erase_Block()` and `HAL_NOR_Erase_Chip()`.
- Read the NOR flash CFI (common flash interface) IDs using the function `HAL_NOR_Read_CFI()`. The read information is stored in the `NOR_CFI_TypeDef` structure declared by the function caller.
- You can also control the NOR device by calling the control APIs `HAL_NOR_WriteOperation_Enable()`/ `HAL_NOR_WriteOperation_Disable()` to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function `HAL_NOR_GetState()`



This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

## NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- `NOR_WRITE` : NOR memory write data to specified address

### 37.2.2 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\_DeInit\(\)`](#)
- [`HAL\_NOR\_MspInit\(\)`](#)
- [`HAL\_NOR\_MspDeInit\(\)`](#)
- [`HAL\_NOR\_MspWait\(\)`](#)

### 37.2.3 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\(\)`](#)

### 37.2.4 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\(\)`](#)

### 37.2.5 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\(\)`](#)

### 37.2.6 HAL\_NOR\_Init

Function Name	<code>HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef * Timing, FMC_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 a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>Timing</b>: pointer to NOR control timing structure</li> <li>• <b>ExtTiming</b>: pointer to NOR extended mode timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.7 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>

### 37.2.8 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>

### 37.2.9 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>

### 37.2.10 HAL\_NOR\_MspWait

Function Name	<b>void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)</b>
Function Description	NOR MSP 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>

### 37.2.11 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 a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li>• <b>pNOR_ID</b>: : pointer to NOR ID structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 37.2.12 HAL\_NOR\_ReturnToReadMode

Function Name **HAL\_StatusTypeDef HAL\_NOR\_ReturnToReadMode (NOR\_HandleTypeDef \* hnor)**

Function Description Returns the NOR memory to Read mode.

Parameters

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- HAL status

### 37.2.13 HAL\_NOR\_Read

Function Name **HAL\_StatusTypeDef HAL\_NOR\_Read (NOR\_HandleTypeDef \* hnor, uint32\_t \* pAddress, uint16\_t \* pData)**

Function Description Read data from NOR memory.

Parameters

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress**: pointer to Device address
- **pData**: : pointer to read data

Return values

- HAL status

### 37.2.14 HAL\_NOR\_Program

Function Name **HAL\_StatusTypeDef HAL\_NOR\_Program (NOR\_HandleTypeDef \* hnor, uint32\_t \* pAddress, uint16\_t \* pData)**

Function Description Program data to NOR memory.

Parameters

- **hnor**: pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress**: Device address
- **pData**: : pointer to the data to write

Return values

- HAL status

### 37.2.15 HAL\_NOR\_ReadBuffer

Function Name **HAL\_StatusTypeDef HAL\_NOR\_ReadBuffer (NOR\_HandleTypeDef \* hnor, uint32\_t uwAddress, uint16\_t \* pData, uint32\_t uwBufferSize)**

Function Description Reads a half-word buffer from the NOR memory.

Parameters

- **hnor**: pointer to the NOR handle
- **uwAddress**: NOR memory internal address to read from.
- **pData**: pointer to the buffer that receives the data read from the NOR memory.

- **uwBufferSize:** : number of Half word to read.
- HAL status

Return values

### 37.2.16 HAL\_NOR\_ProgramBuffer

**Function Name** **HAL\_StatusTypeDef HAL\_NOR\_ProgramBuffer**  
(NOR\_HandleTypeDef \* hnor, uint32\_t uwAddress, uint16\_t \* pData, uint32\_t uwBufferSize)

**Function Description** Writes a half-word buffer to the NOR memory.

- Parameters**
- **hnor:** pointer to the NOR handle
  - **uwAddress:** NOR memory internal start write address
  - **pData:** pointer to source data buffer.
  - **uwBufferSize:** Size of the buffer to write

- Return values**
- HAL status

### 37.2.17 HAL\_NOR\_Erase\_Block

**Function Name** **HAL\_StatusTypeDef HAL\_NOR\_Erase\_Block**  
(NOR\_HandleTypeDef \* hnor, uint32\_t BlockAddress, uint32\_t Address)

**Function Description** Erase the specified block of the NOR memory.

- Parameters**
- **hnor:** pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
  - **BlockAddress:** : Block to erase address
  - **Address:** Device address

- Return values**
- HAL status

### 37.2.18 HAL\_NOR\_Erase\_Chip

**Function Name** **HAL\_StatusTypeDef HAL\_NOR\_Erase\_Chip**  
(NOR\_HandleTypeDef \* hnor, uint32\_t Address)

**Function Description** Erase the entire NOR chip.

- Parameters**
- **hnor:** pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
  - **Address:** : Device address

- Return values**
- HAL status

### 37.2.19 HAL\_NOR\_Read\_CFI

**Function Name** **HAL\_StatusTypeDef HAL\_NOR\_Read\_CFI**  
(NOR\_HandleTypeDef \* hnor, NOR\_CFITypeDef \* pNOR\_CFI)

**Function Description** Read NOR flash CFI IDs.

- Parameters**
- **hnor:** pointer to a NOR\_HandleTypeDef structure that contains the configuration information for NOR module.
  - **pNOR\_CFI:** : pointer to NOR CFI IDs structure

- Return values**
- HAL status

**37.2.20 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 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>

**37.2.21 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 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>

**37.2.22 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 a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>NOR controller state</li> </ul>

**37.2.23 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 a NOR_HandleTypeDef structure that contains the configuration information for NOR module.</li> <li><b>Address:</b> Device address</li> <li><b>Timeout:</b> NOR 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>

**37.3 NOR Firmware driver defines****37.3.1 NOR***NOR Exported Macros*

---

`__HAL_NOR_RESET_HANDLE_STATE`**Description:**

- Reset NOR handle state.

**Parameters:**

- `__HANDLE__`: specifies the NOR handle.

**Return value:**

- None

***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

## 38 HAL PCD Generic Driver

### 38.1 PCD Firmware driver registers structures

#### 38.1.1 PCD\_HandleTypeDef

##### Data Fields

- *PCD\_TypeDef \* Instance*
- *PCD\_InitTypeDef Init*
- *PCD\_EPTTypeDef IN\_ep*
- *PCD\_EPTTypeDef OUT\_ep*
- *HAL\_LockTypeDef Lock*
- *\_\_IO PCD\_StateTypeDef State*
- *uint32\_t Setup*
- *PCD\_LPM\_StateTypeDef LPM\_State*
- *uint32\_t BESL*
- *uint32\_t lpm\_active*
- *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
- *PCD\_LPM\_StateTypeDef PCD\_HandleTypeDef::LPM\_State*  
LPM State
- *uint32\_t PCD\_HandleTypeDef::BESL*
- *uint32\_t PCD\_HandleTypeDef::lpm\_active*  
Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- *void\* PCD\_HandleTypeDef::pData*  
Pointer to upper stack Handler

## 38.2 PCD Firmware driver API description

### 38.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD\_HandleTypeDef handle structure, for example: PCD\_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL\_PCD\_Init() API to initialize the HCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL\_PCD\_MspInit() API:
  - a. Enable the PCD/USB Low Level interface clock using
    - \_\_OTGFS-OTG\_CLK\_ENABLE()/\_\_OTGHS-OTG\_CLK\_ENABLE();
    - \_\_OTGHSULPI\_CLK\_ENABLE(); (For High Speed Mode)
  - b. Initialize the related GPIO clocks
  - c. Configure PCD pin-out
  - d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
  - a. hpcd.pData = pdev;
6. Enable HCD transmission and reception:
  - a. HAL\_PCD\_Start();

### 38.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

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

### 38.2.3 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\(\)\*](#)

### 38.2.4 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\(\)](#)

### 38.2.5 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\(\)](#)

### 38.2.6 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 create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 38.2.7 HAL\_PCD\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DeInit (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>

### 38.2.8 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>

### 38.2.9 HAL\_PCD\_MspDeInit

Function Name	<b>void HAL_PCD_MspDeInit (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>
<b>38.2.10</b>	<b>HAL_PCD_Start</b>	
	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>
<b>38.2.11</b>	<b>HAL_PCD_Stop</b>	
	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>
<b>38.2.12</b>	<b>HAL_PCD_IRQHandler</b>	
	Function Name	<b>void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)</b>
	Function Description	This function handles PCD interrupt request.
	Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
	Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>38.2.13</b>	<b>HAL_PCD_DataOutStageCallback</b>	
	Function Name	<b>void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
	Function Description	Data out stage callbacks.
	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>
<b>38.2.14</b>	<b>HAL_PCD_DataInStageCallback</b>	
	Function Name	<b>void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
	Function Description	Data IN stage callbacks.
	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>

**38.2.15 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>

**38.2.16 HAL\_PCD\_SOFCallback**

Function Name	<b>void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	USB Start Of Frame callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**38.2.17 HAL\_PCD\_ResetCallback**

Function Name	<b>void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	USB Reset callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**38.2.18 HAL\_PCD\_SuspendCallback**

Function Name	<b>void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	Suspend event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**38.2.19 HAL\_PCD\_ResumeCallback**

Function Name	<b>void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)</b>
Function Description	Resume event callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**38.2.20 HAL\_PCD\_ISOOUTIncompleteCallback**

Function Name	<b>void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)</b>
Function Description	Incomplete ISO OUT callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li><li>• <b>epnum</b>: endpoint number</li></ul>

Return values

- None

### 38.2.21 HAL\_PCD\_ISOINIncompleteCallback

Function Name **void HAL\_PCD\_ISOINIncompleteCallback (PCD\_HandleTypeDef \* hpcd, uint8\_t epnum)**

Function Description Incomplete ISO IN callbacks.

Parameters

- **hpcd**: PCD handle
- **epnum**: endpoint number

Return values

- None

### 38.2.22 HAL\_PCD\_ConnectCallback

Function Name **void HAL\_PCD\_ConnectCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description Connection event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

### 38.2.23 HAL\_PCD\_DisconnectCallback

Function Name **void HAL\_PCD\_DisconnectCallback (PCD\_HandleTypeDef \* hpcd)**

Function Description Disconnection event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

### 38.2.24 HAL\_PCD\_DevConnect

Function Name **HAL\_StatusTypeDef HAL\_PCD\_DevConnect (PCD\_HandleTypeDef \* hpcd)**

Function Description Connect the USB device.

Parameters

- **hpcd**: PCD handle

Return values

- HAL status

### 38.2.25 HAL\_PCD\_DevDisconnect

Function Name **HAL\_StatusTypeDef HAL\_PCD\_DevDisconnect (PCD\_HandleTypeDef \* hpcd)**

Function Description Disconnect the USB device.

Parameters

- **hpcd**: PCD handle

Return values

- HAL status

### 38.2.26 HAL\_PCD\_SetAddress

Function Name **HAL\_StatusTypeDef HAL\_PCD\_SetAddress**

---

**(PCD\_HandleTypeDef \* hpcd, uint8\_t address)**

Function Description    Set the USB Device address.

Parameters

- **hpcd**: PCD handle
- **address**: new device address

Return values

- HAL status

### 38.2.27 HAL\_PCD\_EP\_Open

Function Name    **HAL\_StatusTypeDef HAL\_PCD\_EP\_Open**  
**(PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr, uint16\_t**  
**ep\_mps, uint8\_t ep\_type)**

Function Description    Open and configure an endpoint.

Parameters

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address
- **ep\_mps**: endpoint max packet size
- **ep\_type**: endpoint type

Return values

- HAL status

### 38.2.28 HAL\_PCD\_EP\_Close

Function Name    **HAL\_StatusTypeDef HAL\_PCD\_EP\_Close**  
**(PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr)**

Function Description    Deactivate an endpoint.

Parameters

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address

Return values

- HAL status

### 38.2.29 HAL\_PCD\_EP\_Receive

Function Name    **HAL\_StatusTypeDef HAL\_PCD\_EP\_Receive**  
**(PCD\_HandleTypeDef \* hpcd, uint8\_t ep\_addr, uint8\_t \* pBuf,**  
**uint32\_t len)**

Function Description    Receive an amount of data.

Parameters

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address
- **pBuf**: pointer to the reception buffer
- **len**: amount of data to be received

Return values

- HAL status

### 38.2.30 HAL\_PCD\_EP\_GetRxCount

Function Name    **uint16\_t HAL\_PCD\_EP\_GetRxCount (PCD\_HandleTypeDef \***  
**hpcd, uint8\_t ep\_addr)**

Function Description    Get Received Data Size.

Parameters

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address



Return values

- Data Size

### 38.2.31 HAL\_PCD\_EP\_Transmit

**Function Name** `HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)`

**Function Description** Send an amount of data.

**Parameters**

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address
- **pBuf**: pointer to the transmission buffer
- **len**: amount of data to be sent

**Return values**

- HAL status

### 38.2.32 HAL\_PCD\_EP\_SetStall

**Function Name** `HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)`

**Function Description** Set a STALL condition over an endpoint.

**Parameters**

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address

**Return values**

- HAL status

### 38.2.33 HAL\_PCD\_EP\_ClrStall

**Function Name** `HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)`

**Function Description** Clear a STALL condition over in an endpoint.

**Parameters**

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address

**Return values**

- HAL status

### 38.2.34 HAL\_PCD\_EP\_Flush

**Function Name** `HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)`

**Function Description** Flush an endpoint.

**Parameters**

- **hpcd**: PCD handle
- **ep\_addr**: endpoint address

**Return values**

- HAL status

### 38.2.35 HAL\_PCD\_ActivateRemoteWakeup

**Function Name** `HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)`

**Function Description** `HAL_PCD_ActivateRemoteWakeup` : Active remote wake-up 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>

### 38.2.36 HAL\_PCD\_DeActivateRemoteWakeup

Function Name	<b>HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)</b>
Function Description	HAL_PCD_DeActivateRemoteWakeup : de-active remote wake-up 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>

### 38.2.37 HAL\_PCD\_GetState

Function Name	<b>PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)</b>
Function Description	Return the PCD state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

## 38.3 PCD Firmware driver defines

### 38.3.1 PCD

#### *PCD Exported Macros*

```

__HAL_PCD_ENABLE
__HAL_PCD_DISABLE
__HAL_PCD_GET_FLAG
__HAL_PCD_CLEAR_FLAG
__HAL_PCD_IS_INVALID_INTERRUPT
__HAL_PCD_UNGATE_PHYCLOCK
__HAL_PCD_GATE_PHYCLOCK
__HAL_PCD_IS_PHY_SUSPENDED
USB_OTG_FS_WAKEUP_EXTI_RISING_EDGE
USB_OTG_FS_WAKEUP_EXTI_FALLING_EDGE
USB_OTG_FS_WAKEUP_EXTI_RISING_FALLING_EDGE
USB_OTG_HS_WAKEUP_EXTI_RISING_EDGE
USB_OTG_HS_WAKEUP_EXTI_FALLING_EDGE
USB_OTG_HS_WAKEUP_EXTI_RISING_FALLING_EDGE
USB_OTG_HS_WAKEUP_EXTI_LINE

```

External  
interrupt  
line 20  
Connecte

USB\_OTG\_FS\_WAKEUP\_EXTI\_LINE

d to the  
USB HS  
EXTI Line

External  
interrupt  
line 18  
Connecte  
d to the  
USB FS  
EXTI Line

\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_ENABLE\_IT

\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_DISABLE\_IT

\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_GET\_FLAG

\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_CLEAR\_FLAG

\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_ENABLE\_RISING\_EDGE

\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_ENABLE\_FALLING\_EDGE

\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_ENABLE\_RISING\_FALLING\_EDGE

\_\_HAL\_USB\_OTG\_HS\_WAKEUP\_EXTI\_GENERATE\_SWIT

\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_ENABLE\_IT

\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_DISABLE\_IT

\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_GET\_FLAG

\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_CLEAR\_FLAG

\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_ENABLE\_RISING\_EDGE

\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_ENABLE\_FALLING\_EDGE

\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_ENABLE\_RISING\_FALLING\_EDGE

\_\_HAL\_USB\_OTG\_FS\_WAKEUP\_EXTI\_GENERATE\_SWIT

#### **PCD Instance definition**

IS\_PCD\_ALL\_INSTANCE

#### **PCD PHY Module**

PCD\_PHY\_ULPI

PCD\_PHY\_EMBEDDED

#### **PCD 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

## 39 HAL PCD Extension Driver

### 39.1 PCDEx Firmware driver API description

#### 39.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- [HAL\\_PCDEx\\_SetTxFiFo\(\)](#)
- [HAL\\_PCDEx\\_SetRxFiFo\(\)](#)
- [HAL\\_PCDEx\\_ActivateLPM\(\)](#)
- [HAL\\_PCDEx\\_DeActivateLPM\(\)](#)
- [HAL\\_PCDEx\\_LPM\\_Callback\(\)](#)

#### 39.1.2 HAL\_PCDEx\_SetTxFiFo

Function Name	HAL_StatusTypeDef HAL_PCDEx_SetTxFiFo (PCD_HandleTypeDef * hpcd, uint8_t fifo, uint16_t size)
Function Description	Set Tx FIFO.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> <li>• <b>fifo</b>: The number of Tx fifo</li> <li>• <b>size</b>: Fifo size</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 39.1.3 HAL\_PCDEx\_SetRxFiFo

Function Name	HAL_StatusTypeDef HAL_PCDEx_SetRxFiFo (PCD_HandleTypeDef * hpcd, uint16_t size)
Function Description	Set Rx FIFO.
Parameters	<ul style="list-style-type: none"> <li>• <b>hpcd</b>: PCD handle</li> <li>• <b>size</b>: Size of Rx fifo</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 39.1.4 HAL\_PCDEx\_ActivateLPM

Function Name	HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)
Function Description	HAL_PCDEx_ActivateLPM : active LPM Feature.
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>

#### 39.1.5 HAL\_PCDEx\_DeActivateLPM

Function Name	HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM
---------------	---

---

**(PCD\_HandleTypeDef \* hpcd)**

Function Description	HAL_PCDEx_DeActivateLPM : de-active LPM feature.
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>

### 39.1.6 HAL\_PCDEx\_LPM\_Callback

Function Name	<b>void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)</b>
Function Description	HAL_PCDEx_LPM_Callback : Send LPM message to user layer.
Parameters	<ul style="list-style-type: none"><li>• <b>hpcd</b>: PCD handle</li><li>• <b>msg</b>: LPM message</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 40 HAL PWR Generic Driver

### 40.1 PWR Firmware driver registers structures

#### 40.1.1 PWR\_PVDTypeDef

##### Data Fields

- *uint32\_t PVDLevel*
- *uint32\_t Mode*

##### Field Documentation

- *uint32\_t PWR\_PVDTypeDef::PVDLevel*  
PVDLevel: Specifies the PVD detection level. This parameter can be a value of [PWR\\_PVD\\_detection\\_level](#)
- *uint32\_t PWR\_PVDTypeDef::Mode*  
Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [PWR\\_PVD\\_Mode](#)

### 40.2 PWR Firmware driver API description

#### 40.2.1 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\(\)](#)

#### 40.2.2 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 are 6 Wake-up pin in the STM32F7 devices family

## Low Power modes configuration

The devices feature 3 low-power modes:

- Sleep mode: Cortex-M7 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 STM32F7 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-M7 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 or falling edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.

### Auto-wakeup (AWU) from low-power mode

- The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event or a time-stamp event, without depending on an external interrupt (Auto-wakeup mode).
- RTC auto-wakeup (AWU) from the Stop and Standby modes
  - To wake up from the Stop mode with an RTC alarm event, it is necessary to configure the RTC to generate the RTC alarm using the HAL\_RTC\_SetAlarm\_IT() function.
  - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to configure the RTC to detect the tamper or time stamp event using the HAL\_RTCEX\_SetTimeStamp\_IT() or HAL\_RTCEX\_SetTamper\_IT() functions.
  - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to configure the RTC to generate the RTC WakeUp event using the HAL\_RTCEX\_SetWakeUpTimer\_IT() function.

This section contains the following APIs:

- [HAL\\_PWR\\_ConfigPVD\(\)](#)
- [HAL\\_PWR\\_EnablePVD\(\)](#)
- [HAL\\_PWR\\_DisablePVD\(\)](#)
- [HAL\\_PWR\\_EnableWakeUpPin\(\)](#)
- [HAL\\_PWR\\_DisableWakeUpPin\(\)](#)
- [HAL\\_PWR\\_EnterSLEEPMode\(\)](#)
- [HAL\\_PWR\\_EnterSTOPMode\(\)](#)
- [HAL\\_PWR\\_EnterSTANDBYMode\(\)](#)
- [HAL\\_PWR\\_PVD\\_IRQHandler\(\)](#)
- [HAL\\_PWR\\_PVDCallback\(\)](#)
- [HAL\\_PWR\\_EnableSleepOnExit\(\)](#)
- [HAL\\_PWR\\_DisableSleepOnExit\(\)](#)
- [HAL\\_PWR\\_EnableSEVOnPend\(\)](#)
- [HAL\\_PWR\\_DisableSEVOnPend\(\)](#)

#### 40.2.3 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>

#### 40.2.4 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

- If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.

## 40.2.5 HAL\_PWR\_DisableBkUpAccess

## Function Name

**void HAL\_PWR\_DisableBkUpAccess (void )**

## Function Description

Disables access to the backup domain (RTC registers, RTC backup data registers and backup SRAM).

## Return values

- None

## Notes

- If the HSE divided by 2, 3, ..31 is used as the RTC clock, the Backup Domain Access should be kept enabled.

## 40.2.6 HAL\_PWR\_ConfigPVD

## Function Name

**void HAL\_PWR\_ConfigPVD (PWR\_PVTypeDef \* sConfigPVD)**

## Function Description

Configures the voltage threshold detected by the Power Voltage Detector(PVD).

## Parameters

- **sConfigPVD:** pointer to an PWR\_PVTypeDef structure that contains the configuration information for the PVD.

## Return values

- None

## Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

## 40.2.7 HAL\_PWR\_EnablePVD

## Function Name

**void HAL\_PWR\_EnablePVD (void )**

## Function Description

Enables the Power Voltage Detector(PVD).

## Return values

- None

## 40.2.8 HAL\_PWR\_DisablePVD

## Function Name

**void HAL\_PWR\_DisablePVD (void )**

## Function Description

Disables the Power Voltage Detector(PVD).

## Return values

- None

## 40.2.9 HAL\_PWR\_EnableWakeUpPin

## Function Name

**void HAL\_PWR\_EnableWakeUpPin (uint32\_t WakeUpPinPolarity)**

## Function Description

Enable the WakeUp PINx functionality.

## Parameters

- **WakeUpPinPolarity:** Specifies which Wake-Up pin to enable. This parameter can be one of the following legacy values, which sets the default polarity: detection on high level (rising edge): PWR\_WAKEUP\_PIN1, PWR\_WAKEUP\_PIN2, PWR\_WAKEUP\_PIN3, PWR\_WAKEUP\_PIN4, PWR\_WAKEUP\_PIN5, PWR\_WAKEUP\_PIN6 or one of the following value where the user can explicitly states the

enabled pin and the chosen polarity  
PWR\_WAKEUP\_PIN1\_HIGH or  
PWR\_WAKEUP\_PIN1\_LOWPWR\_WAKEUP\_PIN2\_HIGH or  
PWR\_WAKEUP\_PIN2\_LOWPWR\_WAKEUP\_PIN3\_HIGH or  
PWR\_WAKEUP\_PIN3\_LOWPWR\_WAKEUP\_PIN4\_HIGH or  
PWR\_WAKEUP\_PIN4\_LOWPWR\_WAKEUP\_PIN5\_HIGH or  
PWR\_WAKEUP\_PIN5\_LOWPWR\_WAKEUP\_PIN6\_HIGH or  
PWR\_WAKEUP\_PIN6\_LOW

- Return values
- None
- Notes
- PWR\_WAKEUP\_PINx and PWR\_WAKEUP\_PINx\_HIGH are equivalent.

#### 40.2.10 HAL\_PWR\_DisableWakeUpPin

Function Name **void HAL\_PWR\_DisableWakeUpPin (uint32\_t WakeUpPinx)**

Function Description Disables the WakeUp PINx functionality.

- Parameters
- **WakeUpPinx:** Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:  
PWR\_WAKEUP\_PIN1PWR\_WAKEUP\_PIN2PWR\_WAKEUP\_PIN3PWR\_WAKEUP\_PIN4PWR\_WAKEUP\_PIN5PWR\_WAKEUP\_PIN6

- Return values
- None

#### 40.2.11 HAL\_PWR\_EnterSLEEPMode

Function Name **void HAL\_PWR\_EnterSLEEPMode (uint32\_t Regulator, uint8\_t SLEEPEntry)**

Function Description Enters Sleep mode.

- Parameters
- **Regulator:** Specifies the regulator state in SLEEP mode. This parameter can be one of the following values:  
PWR\_MAINREGULATOR\_ON: SLEEP mode with regulator ON  
PWR\_LOWPOWERREGULATOR\_ON: SLEEP mode with low power regulator ON
  - **SLEEPEntry:** Specifies if SLEEP mode is entered with WFI or WFE instruction. This parameter can be one of the following values: PWR\_SLEEPENTRY\_WFI: enter SLEEP mode with WFI instructionPWR\_SLEEPENTRY\_WFE: enter SLEEP mode with WFE instruction

- Return values
- None

- Notes
- In Sleep mode, all I/O pins keep the same state as in Run mode.
  - In Sleep mode, the systick is stopped to avoid exit from this mode with systick interrupt when used as time base for Timeout
  - This parameter is not used for the STM32F7 family and is

kept as parameter just to maintain compatibility with the lower power families.

#### 40.2.12 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 ON PWR_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> <li>• When exiting Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.</li> <li>• When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.</li> </ul>

#### 40.2.13 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 pins if enabled.</li> </ul>

#### 40.2.14 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>

#### 40.2.15 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>

#### 40.2.16 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>

#### 40.2.17 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	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.</li> </ul>

#### 40.2.18 HAL\_PWR\_EnableSEVOnPend

Function Name	<b>void HAL_PWR_EnableSEVOnPend (void )</b>
Function Description	Enables CORTEX M4 SEVONPEND bit.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>

#### 40.2.19 HAL\_PWR\_DisableSEVOnPend

Function Name	<b>void HAL_PWR_DisableSEVOnPend (void )</b>
Function Description	Disables CORTEX M4 SEVONPEND bit.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>

## 40.3 PWR Firmware driver defines

### 40.3.1 PWR

*PWR Enable WUP Mask*

`PWR_EWUP_MASK`

*PWR Exported Macro*

`__HAL_PWR_VOLTAGESCALING_CONFIG`

**Description:**

- macros configure the main internal regulator output voltage.

**Parameters:**

- `__REGULATOR__`: specifies the regulator output voltage to achieve a tradeoff between performance and power consumption when the device does not operate at the maximum frequency (refer to the datasheets for more details). This parameter can be one of the following values:
  - `PWR_REGULATOR_VOLTAGE_SCALE1`: Regulator voltage output Scale 1 mode
  - `PWR_REGULATOR_VOLTAGE_SCALE2`: Regulator voltage output Scale 2 mode
  - `PWR_REGULATOR_VOLTAGE_SCALE3`: Regulator voltage output Scale 3 mode

**Return value:**

- None

**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 on the internal wakeup line in standby mode (RTC alarm (Alarm A or Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup)).
  - `PWR_FLAG_SB`: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
  - `PWR_FLAG_PVDO`: PVD

`__HAL_PWR_GET_FLAG`

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.
- PWR\_FLAG\_VOSRDY: This flag indicates that the Regulator voltage scaling output selection is ready.

**Return value:**

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

**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\_SB: StandBy flag

**Description:**

- Enable the PVD EXTI Line 16.

**Return value:**

- None.

**Description:**

- Disable the PVD EXTI Line 16.

**Return value:**

- None.

**Description:**

- Enable event on PVD EXTI Line 16.

**Return value:**

- None.

**Description:**

- Disable event on PVD EXTI Line 16.

**Return value:**

\_\_HAL\_PWR\_CLEAR\_FLAG

\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_IT

\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_IT

\_\_HAL\_PWR\_PVD\_EXTI\_ENABLE\_EVENT

\_\_HAL\_PWR\_PVD\_EXTI\_DISABLE\_EVENT

`__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE`

- None.

**Description:**

- Enable the PVD Extended Interrupt Rising Trigger.

**Return value:**

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE`

**Description:**

- Disable the PVD Extended Interrupt Rising Trigger.

**Return value:**

- None.

`__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE`

**Description:**

- Enable the PVD Extended Interrupt Falling Trigger.

**Return value:**

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE`

**Description:**

- Disable the PVD Extended Interrupt Falling Trigger.

**Return value:**

- None.

`__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE`

**Description:**

- PVD EXTI line configuration: set rising & falling edge trigger.

**Return value:**

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE`

**Description:**

- Disable the PVD Extended Interrupt Rising & Falling Trigger.

**Return value:**

- None.

`__HAL_PWR_PVD_EXTI_GET_FLAG`

**Description:**

- 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_FLAG_VOSRDY`

***PWR Private macros to check input parameters***

`IS_PWR_WAKEUP_POLARITY`

`IS_PWR_PVD_LEVEL`

`IS_PWR_PVD_MODE`

`IS_PWR_REGULATOR`

`IS_PWR_SLEEP_ENTRY`

`IS_PWR_STOP_ENTRY`

`IS_PWR_REGULATOR_VOLTAGE`

***PWR PVD detection level***

`PWR_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 Regulator state in SLEEP/STOP mode**

PWR\_MAINREGULATOR\_ON

PWR\_LOWPOWERREGULATOR\_ON

**PWR Regulator Voltage Scale**

PWR\_REGULATOR\_VOLTAGE\_SCALE1

PWR\_REGULATOR\_VOLTAGE\_SCALE2

PWR\_REGULATOR\_VOLTAGE\_SCALE3

**PWR SLEEP mode entry**

PWR\_SLEEPENTRY\_WFI

PWR\_SLEEPENTRY\_WFE

**PWR STOP mode entry**

PWR\_STOPENTRY\_WFI

PWR\_STOPENTRY\_WFE

## 41 HAL PWR Extension Driver

### 41.1 PWREx Firmware driver API description

#### 41.1.1 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.
- The main internal regulator can be configured to have a tradeoff between performance and power consumption when the device does not operate at the maximum frequency. This is done through `__HAL_PWR_MAINREGULATORMODE_CONFIG()` macro which configure VOS bit in `PWR_CR` register. Refer to the product datasheets for more details.

##### FLASH Power Down configuration

- By setting the `FPDS` bit in the `PWR_CR` register by using the `HAL_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.

##### Over-Drive and Under-Drive configuration

- In Run mode: the main regulator has 2 operating modes available:
  - Normal mode: The CPU and core logic operate at maximum frequency at a given voltage scaling (scale 1, scale 2 or scale 3)
  - Over-drive mode: This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale 2 or scale 3). This mode is enabled through `HAL_PWREx_EnableOverDrive()` function and disabled by `HAL_PWREx_DisableOverDrive()` function, to enter or exit from Over-drive mode please follow the sequence described in Reference manual.

- In Stop mode: the main regulator or low power regulator supplies a low power voltage to the 1.2V domain, thus preserving the content of registers and internal SRAM. 2 operating modes are available:
  - Normal mode: the 1.2V domain is preserved in nominal leakage mode. This mode is only available when the main regulator or the low power regulator is used in Scale 3 or low voltage mode.
  - Under-drive mode: the 1.2V domain is preserved in reduced leakage mode. This mode is only available when the main regulator or the low power regulator is in low voltage mode.

This section contains the following APIs:

- [\*HAL\\_PWREx\\_EnableBkUpReg\(\)\*](#)
- [\*HAL\\_PWREx\\_DisableBkUpReg\(\)\*](#)
- [\*HAL\\_PWREx\\_EnableFlashPowerDown\(\)\*](#)
- [\*HAL\\_PWREx\\_DisableFlashPowerDown\(\)\*](#)
- [\*HAL\\_PWREx\\_EnableMainRegulatorLowVoltage\(\)\*](#)
- [\*HAL\\_PWREx\\_DisableMainRegulatorLowVoltage\(\)\*](#)
- [\*HAL\\_PWREx\\_EnableLowRegulatorLowVoltage\(\)\*](#)
- [\*HAL\\_PWREx\\_DisableLowRegulatorLowVoltage\(\)\*](#)
- [\*HAL\\_PWREx\\_EnableOverDrive\(\)\*](#)
- [\*HAL\\_PWREx\\_DisableOverDrive\(\)\*](#)
- [\*HAL\\_PWREx\\_EnterUnderDriveSTOPMode\(\)\*](#)
- [\*HAL\\_PWREx\\_GetVoltageRange\(\)\*](#)
- [\*HAL\\_PWREx\\_ControlVoltageScaling\(\)\*](#)

#### 41.1.2 HAL\_PWREx\_EnableBkUpReg

Function Name            **HAL\_StatusTypeDef HAL\_PWREx\_EnableBkUpReg (void )**  
 Function Description    Enables the Backup Regulator.  
 Return values            • HAL status

#### 41.1.3 HAL\_PWREx\_DisableBkUpReg

Function Name            **HAL\_StatusTypeDef HAL\_PWREx\_DisableBkUpReg (void )**  
 Function Description    Disables the Backup Regulator.  
 Return values            • HAL status

#### 41.1.4 HAL\_PWREx\_EnableFlashPowerDown

Function Name            **void HAL\_PWREx\_EnableFlashPowerDown (void )**  
 Function Description    Enables the Flash Power Down in Stop mode.  
 Return values            • None

#### 41.1.5 HAL\_PWREx\_DisableFlashPowerDown

Function Name            **void HAL\_PWREx\_DisableFlashPowerDown (void )**  
 Function Description    Disables the Flash Power Down in Stop mode.  
 Return values            • None

#### 41.1.6 HAL\_PWREx\_EnableMainRegulatorLowVoltage

	Function Name	<b>void HAL_PWREx_EnableMainRegulatorLowVoltage (void )</b>
	Function Description	Enables Main Regulator low voltage mode.
	Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
<b>41.1.7</b>	<b>HAL_PWREx_DisableMainRegulatorLowVoltage</b>	
	Function Name	<b>void HAL_PWREx_DisableMainRegulatorLowVoltage (void )</b>
	Function Description	Disables Main Regulator low voltage mode.
	Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
<b>41.1.8</b>	<b>HAL_PWREx_EnableLowRegulatorLowVoltage</b>	
	Function Name	<b>void HAL_PWREx_EnableLowRegulatorLowVoltage (void )</b>
	Function Description	Enables Low Power Regulator low voltage mode.
	Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
<b>41.1.9</b>	<b>HAL_PWREx_DisableLowRegulatorLowVoltage</b>	
	Function Name	<b>void HAL_PWREx_DisableLowRegulatorLowVoltage (void )</b>
	Function Description	Disables Low Power Regulator low voltage mode.
	Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
<b>41.1.10</b>	<b>HAL_PWREx_EnableOverDrive</b>	
	Function Name	<b>HAL_StatusTypeDef HAL_PWREx_EnableOverDrive (void )</b>
	Function Description	Activates the Over-Drive mode.
	Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
	Notes	<ul style="list-style-type: none"> <li>• This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale 2 or scale 3).</li> <li>• It is recommended to enter or exit Over-drive mode when the application is not running critical tasks and when the system clock source is either HSI or HSE. During the Over-drive switch activation, no peripheral clocks should be enabled. The peripheral clocks must be enabled once the Over-drive mode is activated.</li> </ul>
<b>41.1.11</b>	<b>HAL_PWREx_DisableOverDrive</b>	
	Function Name	<b>HAL_StatusTypeDef HAL_PWREx_DisableOverDrive (void )</b>
	Function Description	Deactivates the Over-Drive mode.
	Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
	Notes	<ul style="list-style-type: none"> <li>• This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for a given voltage scaling (scale 1, scale 2 or scale 3).</li> <li>• It is recommended to enter or exit Over-drive mode when the application is not running critical tasks and when the system</li> </ul>

clock source is either HSI or HSE. During the Over-drive switch activation, no peripheral clocks should be enabled. The peripheral clocks must be enabled once the Over-drive mode is activated.

#### 41.1.12 HAL\_PWREx\_EnterUnderDriveSTOPMode

Function Name	<b>HAL_StatusTypeDef HAL_PWREx_EnterUnderDriveSTOPMode (uint32_t Regulator, uint8_t STOPEntry)</b>
Function Description	Enters in Under-Drive 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_UNDERDRIVE_ON: Main Regulator in under-drive mode and Flash memory in power-down when the device is in Stop under-drive mode PWR_LOWPOWERREGULATOR_UNDERDRIVE_ON: Low Power Regulator in under-drive mode and Flash memory in power-down when the device is in Stop under-drive mode</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_SLEEPENTRY_WFI: enter STOP mode with WFI instruction PWR_SLEEPENTRY_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>• This mode can be selected only when the Under-Drive is already active</li> <li>• This mode is enabled only with STOP low power mode. In this mode, the 1.2V domain is preserved in reduced leakage mode. This mode is only available when the main regulator or the low power regulator is in low voltage mode</li> <li>• If the Under-drive mode was enabled, it is automatically disabled after exiting Stop mode. When the voltage regulator operates in Under-drive mode, an additional startup delay is induced when waking up from Stop mode.</li> <li>• In Stop mode, all I/O pins keep the same state as in Run mode.</li> <li>• When exiting Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock.</li> <li>• When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.</li> </ul>

#### 41.1.13 HAL\_PWREx\_GetVoltageRange

Function Name	<b>uint32_t HAL_PWREx_GetVoltageRange (void )</b>
Function Description	Returns Voltage Scaling Range.

- Return values
- VOS bit field (PWR\_REGULATOR\_VOLTAGE\_SCALE1, PWR\_REGULATOR\_VOLTAGE\_SCALE2 or PWR\_REGULATOR\_VOLTAGE\_SCALE3)PWR\_REGULATOR\_VOLTAGE\_SCALE1

#### 41.1.14 HAL\_PWREx\_ControlVoltageScaling

Function Name **HAL\_StatusTypeDef HAL\_PWREx\_ControlVoltageScaling (uint32\_t VoltageScaling)**

Function Description Configures the main internal regulator output voltage.

Parameters

- VoltageScaling:** specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values: PWR\_REGULATOR\_VOLTAGE\_SCALE1: Regulator voltage output range 1 mode, typical output voltage at 1.4 V, system frequency up to 216 MHz.PWR\_REGULATOR\_VOLTAGE\_SCALE2: Regulator voltage output range 2 mode, typical output voltage at 1.2 V, system frequency up to 180 MHz.PWR\_REGULATOR\_VOLTAGE\_SCALE3: Regulator voltage output range 2 mode, typical output voltage at 1.00 V, system frequency up to 151 MHz.

Return values

- HAL Status

Notes

- To update the system clock frequency(SYSCLK): Set the HSI or HSE as system clock frequency using the HAL\_RCC\_ClockConfig().Call the HAL\_RCC\_OscConfig() to configure the PLL.Call HAL\_PWREx\_ConfigVoltageScaling() API to adjust the voltage scale.Set the new system clock frequency using the HAL\_RCC\_ClockConfig().
- The scale can be modified only when the HSI or HSE clock source is selected as system clock source, otherwise the API returns HAL\_ERROR.
- When the PLL is OFF, the voltage scale 3 is automatically selected and the VOS bits value in the PWR\_CR1 register are not taken in account.
- This API forces the PLL state ON to allow the possibility to configure the voltage scale 1 or 2.
- The new voltage scale is active only when the PLL is ON.

## 41.2 PWREx Firmware driver defines

### 41.2.1 PWREx

#### *PWREx Exported Macro*

\_\_HAL\_PWR\_OVERDRIVE\_ENABLE

\_\_HAL\_PWR\_OVERDRIVE\_DISABLE

\_\_HAL\_PWR\_OVERDRIVESWITCHING\_ENABLE

\_\_HAL\_PWR\_OVERDRIVESWITCHING\_DISABLE

---

`__HAL_PWR_UNDERDRIVE_ENABLE`**Notes:**

- This mode is enabled only with STOP low power mode. In this mode, the 1.2V domain is preserved in reduced leakage mode. This mode is only available when the main regulator or the low power regulator is in low voltage mode. If the Under-drive mode was enabled, it is automatically disabled after exiting Stop mode. When the voltage regulator operates in Under-drive mode, an additional startup delay is induced when waking up from Stop mode.

`__HAL_PWR_UNDERDRIVE_DISABLE``__HAL_PWR_GET_ODRUDR_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_ODRDY`: This flag indicates that the Over-drive mode is ready
  - `PWR_FLAG_ODSWRDY`: This flag indicates that the Over-drive mode switching is ready
  - `PWR_FLAG_UDRDY`: This flag indicates that the Under-drive mode is enabled in Stop mode

**Return value:**

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

`__HAL_PWR_CLEAR_ODRUDR_FLAG``__HAL_PWR_GET_WAKEUP_FLAG`**Description:**

- Check Wake Up flag is set or not.

**Parameters:**

- `__WUFLAG__`: specifies the Wake Up flag to check. This parameter can be one of the following values:
  - `PWR_WAKEUP_PIN_FLAG1`: Wakeup Pin Flag for PA0
  - `PWR_WAKEUP_PIN_FLAG2`: Wakeup Pin Flag for PA2
  - `PWR_WAKEUP_PIN_FLAG3`: Wakeup Pin Flag for PC1
  - `PWR_WAKEUP_PIN_FLAG4`:



`__HAL_PWR_CLEAR_WAKEUP_FLAG`

- Wakeup Pin Flag for PC13
- PWR\_WAKEUP\_PIN\_FLAG5: Wakeup Pin Flag for PI8
- PWR\_WAKEUP\_PIN\_FLAG6: Wakeup Pin Flag for PI11

**Description:**

- Clear the WakeUp pins flags.

**Parameters:**

- `__WUFLAG__`: specifies the Wake Up pin flag to clear. This parameter can be one of the following values:
  - PWR\_WAKEUP\_PIN\_FLAG1: Wakeup Pin Flag for PA0
  - PWR\_WAKEUP\_PIN\_FLAG2: Wakeup Pin Flag for PA2
  - PWR\_WAKEUP\_PIN\_FLAG3: Wakeup Pin Flag for PC1
  - PWR\_WAKEUP\_PIN\_FLAG4: Wakeup Pin Flag for PC13
  - PWR\_WAKEUP\_PIN\_FLAG5: Wakeup Pin Flag for PI8
  - PWR\_WAKEUP\_PIN\_FLAG6: Wakeup Pin Flag for PI11

***PWREx Private macros to check input parameters***

`IS_PWR_REGULATOR_UNDERDRIVE`

`IS_PWR_WAKEUP_PIN`

***PWREx Over Under Drive Flag***

`PWR_FLAG_ODRDY`

`PWR_FLAG_ODSWRDY`

`PWR_FLAG_UDRDY`

***PWREx Private Constants***

`PWR_OVERDRIVE_TIMEOUT_VALUE`

`PWR_UDERDRIVE_TIMEOUT_VALUE`

`PWR_BKPREG_TIMEOUT_VALUE`

`PWR_VOSRDY_TIMEOUT_VALUE`

***PWREx Regulator state in UnderDrive mode***

`PWR_MAINREGULATOR_UNDERDRIVE_ON`

`PWR_LOWPOWERREGULATOR_UNDERDRIVE_ON`

***PWREx Wake Up Pins***

`PWR_WAKEUP_PIN1`

`PWR_WAKEUP_PIN2`

`PWR_WAKEUP_PIN3`

PWR\_WAKEUP\_PIN4  
PWR\_WAKEUP\_PIN5  
PWR\_WAKEUP\_PIN6  
PWR\_WAKEUP\_PIN1\_HIGH  
PWR\_WAKEUP\_PIN2\_HIGH  
PWR\_WAKEUP\_PIN3\_HIGH  
PWR\_WAKEUP\_PIN4\_HIGH  
PWR\_WAKEUP\_PIN5\_HIGH  
PWR\_WAKEUP\_PIN6\_HIGH  
PWR\_WAKEUP\_PIN1\_LOW  
PWR\_WAKEUP\_PIN2\_LOW  
PWR\_WAKEUP\_PIN3\_LOW  
PWR\_WAKEUP\_PIN4\_LOW  
PWR\_WAKEUP\_PIN5\_LOW  
PWR\_WAKEUP\_PIN6\_LOW

***PWREx Wake Up Pin Flags***

PWR\_WAKEUP\_PIN\_FLAG1  
PWR\_WAKEUP\_PIN\_FLAG2  
PWR\_WAKEUP\_PIN\_FLAG3  
PWR\_WAKEUP\_PIN\_FLAG4  
PWR\_WAKEUP\_PIN\_FLAG5  
PWR\_WAKEUP\_PIN\_FLAG6

## 42 HAL QSPI Generic Driver

### 42.1 QSPI Firmware driver registers structures

#### 42.1.1 QSPI\_InitTypeDef

##### Data Fields

- *uint32\_t* *ClockPrescaler*
- *uint32\_t* *FifoThreshold*
- *uint32\_t* *SampleShifting*
- *uint32\_t* *FlashSize*
- *uint32\_t* *ChipSelectHighTime*
- *uint32\_t* *ClockMode*
- *uint32\_t* *FlashID*
- *uint32\_t* *DualFlash*

##### Field Documentation

- *uint32\_t* *QSPI\_InitTypeDef::ClockPrescaler*
- *uint32\_t* *QSPI\_InitTypeDef::FifoThreshold*
- *uint32\_t* *QSPI\_InitTypeDef::SampleShifting*
- *uint32\_t* *QSPI\_InitTypeDef::FlashSize*
- *uint32\_t* *QSPI\_InitTypeDef::ChipSelectHighTime*
- *uint32\_t* *QSPI\_InitTypeDef::ClockMode*
- *uint32\_t* *QSPI\_InitTypeDef::FlashID*
- *uint32\_t* *QSPI\_InitTypeDef::DualFlash*

#### 42.1.2 QSPI\_HandleTypeDef

##### Data Fields

- *QUADSPI\_TypeDef \* Instance*
- *QSPI\_InitTypeDef Init*
- *uint8\_t \* pTxBuffPtr*
- *\_\_IO uint16\_t TxXferSize*
- *\_\_IO uint16\_t TxXferCount*
- *uint8\_t \* pRxBuffPtr*
- *\_\_IO uint16\_t RxXferSize*
- *\_\_IO uint16\_t RxXferCount*
- *DMA\_HandleTypeDef \* hdma*
- *\_\_IO HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_QSPI\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*
- *uint32\_t Timeout*

## Field Documentation

- *QUADSPI\_TypeDef\* QSPI\_HandleTypeDef::Instance*
- *QSPI\_InitTypeDef QSPI\_HandleTypeDef::Init*
- *uint8\_t\* QSPI\_HandleTypeDef::pTxBuffPtr*
- *\_\_IO uint16\_t QSPI\_HandleTypeDef::TxXferSize*
- *\_\_IO uint16\_t QSPI\_HandleTypeDef::TxXferCount*
- *uint8\_t\* QSPI\_HandleTypeDef::pRxBuffPtr*
- *\_\_IO uint16\_t QSPI\_HandleTypeDef::RxXferSize*
- *\_\_IO uint16\_t QSPI\_HandleTypeDef::RxXferCount*
- *DMA\_HandleTypeDef\* QSPI\_HandleTypeDef::hdma*
- *\_\_IO HAL\_LockTypeDef QSPI\_HandleTypeDef::Lock*
- *\_\_IO HAL\_QSPI\_StateTypeDef QSPI\_HandleTypeDef::State*
- *\_\_IO uint32\_t QSPI\_HandleTypeDef::ErrorCode*
- *uint32\_t QSPI\_HandleTypeDef::Timeout*

## 42.1.3 QSPI\_CommandTypeDef

## Data Fields

- *uint32\_t Instruction*
- *uint32\_t Address*
- *uint32\_t AlternateBytes*
- *uint32\_t AddressSize*
- *uint32\_t AlternateBytesSize*
- *uint32\_t DummyCycles*
- *uint32\_t InstructionMode*
- *uint32\_t AddressMode*
- *uint32\_t AlternateByteMode*
- *uint32\_t DataMode*
- *uint32\_t NbData*
- *uint32\_t DdrMode*
- *uint32\_t DdrHoldHalfCycle*
- *uint32\_t SIOOMode*

## Field Documentation

- *uint32\_t QSPI\_CommandTypeDef::Instruction*
- *uint32\_t QSPI\_CommandTypeDef::Address*
- *uint32\_t QSPI\_CommandTypeDef::AlternateBytes*
- *uint32\_t QSPI\_CommandTypeDef::AddressSize*
- *uint32\_t QSPI\_CommandTypeDef::AlternateBytesSize*
- *uint32\_t QSPI\_CommandTypeDef::DummyCycles*
- *uint32\_t QSPI\_CommandTypeDef::InstructionMode*
- *uint32\_t QSPI\_CommandTypeDef::AddressMode*
- *uint32\_t QSPI\_CommandTypeDef::AlternateByteMode*
- *uint32\_t QSPI\_CommandTypeDef::DataMode*
- *uint32\_t QSPI\_CommandTypeDef::NbData*
- *uint32\_t QSPI\_CommandTypeDef::DdrMode*
- *uint32\_t QSPI\_CommandTypeDef::DdrHoldHalfCycle*
- *uint32\_t QSPI\_CommandTypeDef::SIOOMode*

## 42.1.4 QSPI\_AutoPollingTypeDef

### Data Fields

- *uint32\_t Match*
- *uint32\_t Mask*
- *uint32\_t Interval*
- *uint32\_t StatusBytesSize*
- *uint32\_t MatchMode*
- *uint32\_t AutomaticStop*

### Field Documentation

- *uint32\_t QSPI\_AutoPollingTypeDef::Match*
- *uint32\_t QSPI\_AutoPollingTypeDef::Mask*
- *uint32\_t QSPI\_AutoPollingTypeDef::Interval*
- *uint32\_t QSPI\_AutoPollingTypeDef::StatusBytesSize*
- *uint32\_t QSPI\_AutoPollingTypeDef::MatchMode*
- *uint32\_t QSPI\_AutoPollingTypeDef::AutomaticStop*

## 42.1.5 QSPI\_MemoryMappedTypeDef

### Data Fields

- *uint32\_t TimeOutPeriod*
- *uint32\_t TimeOutActivation*

### Field Documentation

- *uint32\_t QSPI\_MemoryMappedTypeDef::TimeOutPeriod*
- *uint32\_t QSPI\_MemoryMappedTypeDef::TimeOutActivation*

## 42.2 QSPI Firmware driver API description

### 42.2.1 How to use this driver

#### Initialization

1. As prerequisite, fill in the HAL\_QSPI\_MspInit() :
  - Enable QuadSPI clock interface with `__HAL_RCC_QSPI_CLK_ENABLE()`.
  - Reset QuadSPI IP with `__HAL_RCC_QSPI_FORCE_RESET()` and `__HAL_RCC_QSPI_RELEASE_RESET()`.
  - Enable the clocks for the QuadSPI GPIOs with `__HAL_RCC_GPIOx_CLK_ENABLE()`.

- Configure these QuadSPI pins in alternate mode using HAL\_GPIO\_Init().
  - If interrupt mode is used, enable and configure QuadSPI global interrupt with HAL\_NVIC\_SetPriority() and HAL\_NVIC\_EnableIRQ().
  - If DMA mode is used, enable the clocks for the QuadSPI DMA channel with \_\_HAL\_RCC\_DMAx\_CLK\_ENABLE(), configure DMA with HAL\_DMA\_Init(), link it with QuadSPI handle using \_\_HAL\_LINKDMA(), enable and configure DMA channel global interrupt with HAL\_NVIC\_SetPriority() and HAL\_NVIC\_EnableIRQ().
2. Configure the flash size, the clock prescaler, the fifo threshold, the clock mode, the sample shifting and the CS high time using the HAL\_QSPI\_Init() function.

### Indirect functional mode

1. Configure the command sequence using the HAL\_QSPI\_Command() or HAL\_QSPI\_Command\_IT() functions :
  - Instruction phase : the mode used and if present the instruction opcode.
  - Address phase : the mode used and if present the size and the address value.
  - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
  - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
  - Data phase : the mode used and if present the number of bytes.
  - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
  - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
2. If no data is required for the command, it is sent directly to the memory :
  - In polling mode, the output of the function is done when the transfer is complete.
  - In interrupt mode, HAL\_QSPI\_CmdCpltCallback() will be called when the transfer is complete.
3. For the indirect write mode, use HAL\_QSPI\_Transmit(), HAL\_QSPI\_Transmit\_DMA() or HAL\_QSPI\_Transmit\_IT() after the command configuration :
  - In polling mode, the output of the function is done when the transfer is complete.
  - In interrupt mode, HAL\_QSPI\_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL\_QSPI\_TxCpltCallback() will be called when the transfer is complete.
  - In DMA mode, HAL\_QSPI\_TxHalfCpltCallback() will be called at the half transfer and HAL\_QSPI\_TxCpltCallback() will be called when the transfer is complete.
4. For the indirect read mode, use HAL\_QSPI\_Receive(), HAL\_QSPI\_Receive\_DMA() or HAL\_QSPI\_Receive\_IT() after the command configuration :
  - In polling mode, the output of the function is done when the transfer is complete.
  - In interrupt mode, HAL\_QSPI\_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL\_QSPI\_RxCpltCallback() will be called when the transfer is complete.
  - In DMA mode, HAL\_QSPI\_RxHalfCpltCallback() will be called at the half transfer and HAL\_QSPI\_RxCpltCallback() will be called when the transfer is complete.

### Auto-polling functional mode

1. Configure the command sequence and the auto-polling functional mode using the HAL\_QSPI\_AutoPolling() or HAL\_QSPI\_AutoPolling\_IT() functions :
  - Instruction phase : the mode used and if present the instruction opcode.

- Address phase : the mode used and if present the size and the address value.
  - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
  - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
  - Data phase : the mode used.
  - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
  - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
  - The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.
2. After the configuration :
    - In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
    - In interrupt mode, HAL\_QSPI\_StatusMatchCallback() will be called each time the status match is reached.

### Memory-mapped functional mode

1. Configure the command sequence and the memory-mapped functional mode using the HAL\_QSPI\_MemoryMapped() functions :
  - Instruction phase : the mode used and if present the instruction opcode.
  - Address phase : the mode used and the size.
  - Alternate-bytes phase : the mode used and if present the size and the alternate bytes values.
  - Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
  - Data phase : the mode used.
  - Double Data Rate (DDR) mode : the activation (or not) of this mode and the delay if activated.
  - Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
  - The timeout activation and the timeout period.
2. After the configuration, the QuadSPI will be used as soon as an access on the AHB is done on the address range. HAL\_QSPI\_TimeOutCallback() will be called when the timeout expires.

### Errors management and abort functionality

1. HAL\_QSPI\_GetError() function gives the error raised during the last operation.
2. HAL\_QSPI\_Abort() function aborts any on-going operation and flushes the fifo.
3. HAL\_QSPI\_GetState() function gives the current state of the HAL QuadSPI driver.

### Workarounds linked to Silicon Limitation

1. Workarounds Implemented inside HAL Driver
  - Extra data written in the FIFO at the end of a read transfer

## 42.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to :

- Initialize the QuadSPI.
- De-initialize the QuadSPI.

This section contains the following APIs:

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

## 42.2.3 IO operation functions

This subsection provides a set of functions allowing to :

- Handle the interrupts.
- Handle the command sequence.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- [\*HAL\\_QSPI\\_IRQHandler\(\)\*](#)
- [\*HAL\\_QSPI\\_Command\(\)\*](#)
- [\*HAL\\_QSPI\\_Command\\_IT\(\)\*](#)
- [\*HAL\\_QSPI\\_Transmit\(\)\*](#)
- [\*HAL\\_QSPI\\_Receive\(\)\*](#)
- [\*HAL\\_QSPI\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_QSPI\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_QSPI\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_QSPI\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_QSPI\\_AutoPolling\(\)\*](#)
- [\*HAL\\_QSPI\\_AutoPolling\\_IT\(\)\*](#)
- [\*HAL\\_QSPI\\_MemoryMapped\(\)\*](#)
- [\*HAL\\_QSPI\\_ErrorCallback\(\)\*](#)
- [\*HAL\\_QSPI\\_CmdCpltCallback\(\)\*](#)
- [\*HAL\\_QSPI\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_QSPI\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_QSPI\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_QSPI\\_TxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_QSPI\\_FifoThresholdCallback\(\)\*](#)
- [\*HAL\\_QSPI\\_StatusMatchCallback\(\)\*](#)
- [\*HAL\\_QSPI\\_TimeOutCallback\(\)\*](#)

## 42.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to :

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation. ....

This section contains the following APIs:



- [HAL\\_QSPI\\_GetState\(\)](#)
- [HAL\\_QSPI\\_GetError\(\)](#)
- [HAL\\_QSPI\\_Abort\(\)](#)
- [HAL\\_QSPI\\_SetTimeout\(\)](#)
- [HAL\\_QSPI\\_ErrorCallback\(\)](#)
- [HAL\\_QSPI\\_FifoThresholdCallback\(\)](#)
- [HAL\\_QSPI\\_CmdCpltCallback\(\)](#)
- [HAL\\_QSPI\\_RxCpltCallback\(\)](#)
- [HAL\\_QSPI\\_TxCpltCallback\(\)](#)
- [HAL\\_QSPI\\_RxHalfCpltCallback\(\)](#)
- [HAL\\_QSPI\\_TxHalfCpltCallback\(\)](#)
- [HAL\\_QSPI\\_StatusMatchCallback\(\)](#)
- [HAL\\_QSPI\\_TimeOutCallback\(\)](#)

#### 42.2.5 HAL\_QSPI\_Init

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Init (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Initializes the QSPI mode according to the specified parameters in the QSPI_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi:</b> qspi handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.6 HAL\_QSPI\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_DeInit (QSPI_HandleTypeDef * hqspi)</b>
Function Description	DeInitializes the QSPI peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi:</b> qspi handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 42.2.7 HAL\_QSPI\_MspInit

Function Name	<b>void HAL_QSPI_MspInit (QSPI_HandleTypeDef * hqspi)</b>
Function Description	QSPI MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi:</b> QSPI handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 42.2.8 HAL\_QSPI\_MspDeInit

Function Name	<b>void HAL_QSPI_MspDeInit (QSPI_HandleTypeDef * hqspi)</b>
Function Description	QSPI MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi:</b> QSPI handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 42.2.9 HAL\_QSPI\_IRQHandler

Function Name	<b>void HAL_QSPI_IRQHandler (QSPI_HandleTypeDef * hqspi)</b>
Function Description	This function handles QSPI interrupt request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None.</li> </ul>

#### 42.2.10 HAL\_QSPI\_Command

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Command (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, uint32_t Timeout)</b>
Function Description	Sets the command configuration.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> <li>• <b>cmd</b>: : structure that contains the command configuration information</li> <li>• <b>Timeout</b>: : Time out duration</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 only in Indirect Read or Write Modes</li> </ul>

#### 42.2.11 HAL\_QSPI\_Command\_IT

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Command_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd)</b>
Function Description	Sets the command configuration in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> <li>• <b>cmd</b>: : structure that contains the command configuration information</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 only in Indirect Read or Write Modes</li> </ul>

#### 42.2.12 HAL\_QSPI\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Transmit (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t Timeout)</b>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> <li>• <b>pData</b>: pointer to data buffer</li> <li>• <b>Timeout</b>: : Time out duration</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 only in Indirect Write Mode</li> </ul>

#### 42.2.13 HAL\_QSPI\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Receive (QSPI_HandleTypeDef * hqspi, uint8_t * pData, uint32_t</b>
---------------	--

**Timeout)**

Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> <li>• <b>pData</b>: pointer to data buffer</li> <li>• <b>Timeout</b>: : Time out duration</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 only in Indirect Read Mode</li> </ul>

**42.2.14 HAL\_QSPI\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Transmit_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)</b>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> <li>• <b>pData</b>: pointer to data buffer</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 only in Indirect Write Mode</li> </ul>

**42.2.15 HAL\_QSPI\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Receive_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)</b>
Function Description	Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> <li>• <b>pData</b>: pointer to data buffer</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 only in Indirect Read Mode</li> </ul>

**42.2.16 HAL\_QSPI\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Transmit_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)</b>
Function Description	Sends an amount of data in non blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> <li>• <b>pData</b>: pointer to data buffer</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 only in Indirect Write Mode</li> </ul>

**42.2.17 HAL\_QSPI\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_QSPI_Receive_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)</b>
Function Description	Receives an amount of data in non blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hqspi</b>: QSPI handle</li> </ul>

- **pData:** pointer to data buffer.
- HAL status
- This function is used only in Indirect Read Mode

#### 42.2.18 HAL\_QSPI\_AutoPolling

- Function Name** `HAL_StatusTypeDef HAL_QSPI_AutoPolling (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)`
- Function Description** Configure the QSPI Automatic Polling Mode in blocking mode.
- Parameters**
- **hqspi:** QSPI handle
  - **cmd:** structure that contains the command configuration information.
  - **cfg:** structure that contains the polling configuration information.
  - **Timeout:** : Time out duration
- Return values**
- HAL status
- Notes**
- This function is used only in Automatic Polling Mode

#### 42.2.19 HAL\_QSPI\_AutoPolling\_IT

- Function Name** `HAL_StatusTypeDef HAL_QSPI_AutoPolling_IT (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg)`
- Function Description** Configure the QSPI Automatic Polling Mode in non-blocking mode.
- Parameters**
- **hqspi:** QSPI handle
  - **cmd:** structure that contains the command configuration information.
  - **cfg:** structure that contains the polling configuration information.
- Return values**
- HAL status
- Notes**
- This function is used only in Automatic Polling Mode

#### 42.2.20 HAL\_QSPI\_MemoryMapped

- Function Name** `HAL_StatusTypeDef HAL_QSPI_MemoryMapped (QSPI_HandleTypeDef * hqspi, QSPI_CommandTypeDef * cmd, QSPI_MemoryMappedTypeDef * cfg)`
- Function Description** Configure the Memory Mapped mode.
- Parameters**
- **hqspi:** QSPI handle
  - **cmd:** structure that contains the command configuration information.
  - **cfg:** structure that contains the memory mapped configuration information.
- Return values**
- HAL status
- Notes**
- This function is used only in Memory mapped Mode

**42.2.21 HAL\_QSPI\_ErrorCallback**

Function Name	<b>void HAL_QSPI_ErrorCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Transfer Error callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**42.2.22 HAL\_QSPI\_CmdCpltCallback**

Function Name	<b>void HAL_QSPI_CmdCpltCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Command completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**42.2.23 HAL\_QSPI\_RxCpltCallback**

Function Name	<b>void HAL_QSPI_RxCpltCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**42.2.24 HAL\_QSPI\_TxCpltCallback**

Function Name	<b>void HAL_QSPI_TxCpltCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**42.2.25 HAL\_QSPI\_RxHalfCpltCallback**

Function Name	<b>void HAL_QSPI_RxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**42.2.26 HAL\_QSPI\_TxHalfCpltCallback**

Function Name	<b>void HAL_QSPI_TxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>

Return values

- None

#### 42.2.27 HAL\_QSPI\_FifoThresholdCallback

Function Name **void HAL\_QSPI\_FifoThresholdCallback (QSPI\_HandleTypeDef \* hqspi)**

Function Description FIFO Threshold callbacks.

Parameters

- **hqspi**: QSPI handle

Return values

- None

#### 42.2.28 HAL\_QSPI\_StatusMatchCallback

Function Name **void HAL\_QSPI\_StatusMatchCallback (QSPI\_HandleTypeDef \* hqspi)**

Function Description Status Match callbacks.

Parameters

- **hqspi**: QSPI handle

Return values

- None

#### 42.2.29 HAL\_QSPI\_TimeOutCallback

Function Name **void HAL\_QSPI\_TimeOutCallback (QSPI\_HandleTypeDef \* hqspi)**

Function Description Timeout callbacks.

Parameters

- **hqspi**: QSPI handle

Return values

- None

#### 42.2.30 HAL\_QSPI\_GetState

Function Name **HAL\_QSPI\_StateTypeDef HAL\_QSPI\_GetState (QSPI\_HandleTypeDef \* hqspi)**

Function Description Return the QSPI state.

Parameters

- **hqspi**: QSPI handle

Return values

- HAL state

#### 42.2.31 HAL\_QSPI\_GetError

Function Name **uint32\_t HAL\_QSPI\_GetError (QSPI\_HandleTypeDef \* hqspi)**

Function Description Return the QSPI error code.

Parameters

- **hqspi**: QSPI handle

Return values

- QSPI Error Code

#### 42.2.32 HAL\_QSPI\_Abort

Function Name **HAL\_StatusTypeDef HAL\_QSPI\_Abort (QSPI\_HandleTypeDef \* hqspi)**

Function Description    Abort the current transmission.

Parameters              •    **hqspi**: QSPI handle

Return values            •    HAL status

#### 42.2.33    **HAL\_QSPI\_SetTimeout**

Function Name            **void HAL\_QSPI\_SetTimeout (QSPI\_HandleTypeDef \* hqspi, uint32\_t Timeout)**

Function Description    Set QSPI timeout.

Parameters              •    **hqspi**: QSPI handle.  
                              •    **Timeout**: Timeout for the QSPI memory access.

Return values            •    None

#### 42.2.34    **HAL\_QSPI\_ErrorCallback**

Function Name            **void HAL\_QSPI\_ErrorCallback (QSPI\_HandleTypeDef \* hqspi)**

Function Description    Transfer Error callbacks.

Parameters              •    **hqspi**: QSPI handle

Return values            •    None

#### 42.2.35    **HAL\_QSPI\_FifoThresholdCallback**

Function Name            **void HAL\_QSPI\_FifoThresholdCallback (QSPI\_HandleTypeDef \* hqspi)**

Function Description    FIFO Threshold callbacks.

Parameters              •    **hqspi**: QSPI handle

Return values            •    None

#### 42.2.36    **HAL\_QSPI\_CmdCpltCallback**

Function Name            **void HAL\_QSPI\_CmdCpltCallback (QSPI\_HandleTypeDef \* hqspi)**

Function Description    Command completed callbacks.

Parameters              •    **hqspi**: QSPI handle

Return values            •    None

#### 42.2.37    **HAL\_QSPI\_RxCpltCallback**

Function Name            **void HAL\_QSPI\_RxCpltCallback (QSPI\_HandleTypeDef \* hqspi)**

Function Description    Rx Transfer completed callbacks.

Parameters              •    **hqspi**: QSPI handle

Return values            •    None

#### 42.2.38    **HAL\_QSPI\_TxCpltCallback**

---

Function Name	<b>void HAL_QSPI_TxCpltCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

#### 42.2.39 HAL\_QSPI\_RxHalfCpltCallback

Function Name	<b>void HAL_QSPI_RxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

#### 42.2.40 HAL\_QSPI\_TxHalfCpltCallback

Function Name	<b>void HAL_QSPI_TxHalfCpltCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

#### 42.2.41 HAL\_QSPI\_StatusMatchCallback

Function Name	<b>void HAL_QSPI_StatusMatchCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Status Match callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

#### 42.2.42 HAL\_QSPI\_TimeOutCallback

Function Name	<b>void HAL_QSPI_TimeOutCallback (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Timeout callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

#### 42.2.43 HAL\_QSPI\_GetState

Function Name	<b>HAL_QSPI_StateTypeDef HAL_QSPI_GetState (QSPI_HandleTypeDef * hqspi)</b>
Function Description	Return the QSPI state.
Parameters	<ul style="list-style-type: none"><li>• <b>hqspi</b>: QSPI handle</li></ul>



Return values

- HAL state

#### 42.2.44 HAL\_QSPI\_GetError

Function Name **uint32\_t HAL\_QSPI\_GetError (QSPI\_HandleTypeDef \* hqspi)**

Function Description Return the QSPI error code.

Parameters

- **hqspi**: QSPI handle

Return values

- QSPI Error Code

#### 42.2.45 HAL\_QSPI\_Abort

Function Name **HAL\_StatusTypeDef HAL\_QSPI\_Abort (QSPI\_HandleTypeDef \* hqspi)**

Function Description Abort the current transmission.

Parameters

- **hqspi**: QSPI handle

Return values

- HAL status

#### 42.2.46 HAL\_QSPI\_SetTimeout

Function Name **void HAL\_QSPI\_SetTimeout (QSPI\_HandleTypeDef \* hqspi, uint32\_t Timeout)**

Function Description Set QSPI timeout.

Parameters

- **hqspi**: QSPI handle.
- **Timeout**: Timeout for the QSPI memory access.

Return values

- None

### 42.3 QSPI Firmware driver defines

#### 42.3.1 QSPI

##### **QSPI Address Mode**

QSPI\_ADDRESS\_NONE No address  
 QSPI\_ADDRESS\_1\_LINE Address on a single line  
 QSPI\_ADDRESS\_2\_LINES Address on two lines  
 QSPI\_ADDRESS\_4\_LINES Address on four lines

##### **QSPI Address Size**

QSPI\_ADDRESS\_8\_BITS 8-bit address  
 QSPI\_ADDRESS\_16\_BITS 16-bit address  
 QSPI\_ADDRESS\_24\_BITS 24-bit address  
 QSPI\_ADDRESS\_32\_BITS 32-bit address

##### **QSPI Alternate Bytes Mode**

QSPI\_ALTERNATE\_BYTES\_NONE No alternate bytes  
 QSPI\_ALTERNATE\_BYTES\_1\_LINE Alternate bytes on a single line

QSPI\_ALTERNATE\_BYTES\_2\_LINES    Alternate bytes on two lines

QSPI\_ALTERNATE\_BYTES\_4\_LINES    Alternate bytes on four lines

#### **QSPI Alternate Bytes Size**

QSPI\_ALTERNATE\_BYTES\_8\_BITS    8-bit alternate bytes

QSPI\_ALTERNATE\_BYTES\_16\_BITS    16-bit alternate bytes

QSPI\_ALTERNATE\_BYTES\_24\_BITS    24-bit alternate bytes

QSPI\_ALTERNATE\_BYTES\_32\_BITS    32-bit alternate bytes

#### **QSPI Automatic Stop**

QSPI\_AUTOMATIC\_STOP\_DISABLE    AutoPolling stops only with abort or QSPI disabling

QSPI\_AUTOMATIC\_STOP\_ENABLE    AutoPolling stops as soon as there is a match

#### **QSPI Chip Select High Time**

QSPI\_CS\_HIGH\_TIME\_1\_CYCLE    nCS stay high for at least 1 clock cycle between commands

QSPI\_CS\_HIGH\_TIME\_2\_CYCLE    nCS stay high for at least 2 clock cycles between commands

QSPI\_CS\_HIGH\_TIME\_3\_CYCLE    nCS stay high for at least 3 clock cycles between commands

QSPI\_CS\_HIGH\_TIME\_4\_CYCLE    nCS stay high for at least 4 clock cycles between commands

QSPI\_CS\_HIGH\_TIME\_5\_CYCLE    nCS stay high for at least 5 clock cycles between commands

QSPI\_CS\_HIGH\_TIME\_6\_CYCLE    nCS stay high for at least 6 clock cycles between commands

QSPI\_CS\_HIGH\_TIME\_7\_CYCLE    nCS stay high for at least 7 clock cycles between commands

QSPI\_CS\_HIGH\_TIME\_8\_CYCLE    nCS stay high for at least 8 clock cycles between commands

#### **QSPI Clock Mode**

QSPI\_CLOCK\_MODE\_0    Clk stays low while nCS is released

QSPI\_CLOCK\_MODE\_3    Clk goes high while nCS is released

#### **QSPI Clock Prescaler**

IS\_QSPI\_CLOCK\_PRESCALER

#### **QSPI Data Mode**

QSPI\_DATA\_NONE    No data

QSPI\_DATA\_1\_LINE    Data on a single line

QSPI\_DATA\_2\_LINES    Data on two lines

QSPI\_DATA\_4\_LINES    Data on four lines

#### **QSPI Ddr HoldHalfCycle**

QSPI_DDR_HHC_ANALOG_DELAY	Delay the data output using analog delay in DDR mode
QSPI_DDR_HHC_HALF_CLK_DELAY	Delay the data output by 1/2 clock cycle in DDR mode

**QSPI Ddr Mode**

QSPI_DDR_MODE_DISABLE	Double data rate mode disabled
QSPI_DDR_MODE_ENABLE	Double data rate mode enabled

**QSPI Dual Flash Mode**

QSPI_DUALFLASH_ENABLE
QSPI_DUALFLASH_DISABLE

**QSPI Dummy Cycles**

IS_QSPI_DUMMY_CYCLES
----------------------

**QSPI Error Code**

HAL_QSPI_ERROR_NONE	No error
HAL_QSPI_ERROR_TIMEOUT	Timeout error
HAL_QSPI_ERROR_TRANSFER	Transfer error
HAL_QSPI_ERROR_DMA	DMA transfer error

**QSPI Exported Macros**

__HAL_QSPI_RESET_HANDLE_STATE	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset QSPI handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: QSPI handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_QSPI_ENABLE	<b>Description:</b> <ul style="list-style-type: none"> <li>Enable QSPI.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: specifies the QSPI Handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_QSPI_DISABLE	<b>Description:</b> <ul style="list-style-type: none"> <li>Disable QSPI.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>__HANDLE__: specifies the QSPI Handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_QSPI_ENABLE_IT	<b>Description:</b>

- Enables the specified QSPI interrupt.

**Parameters:**

- `__HANDLE__`: specifies the QSPI Handle.
- `__INTERRUPT__`: specifies the QSPI interrupt source to enable. This parameter can be one of the following values:
  - `QSPI_IT_TO`: QSPI Time out interrupt
  - `QSPI_IT_SM`: QSPI Status match interrupt
  - `QSPI_IT_FT`: QSPI FIFO threshold interrupt
  - `QSPI_IT_TC`: QSPI Transfer complete interrupt
  - `QSPI_IT_TE`: QSPI Transfer error interrupt

**Return value:**

- None

**Description:**

- Disables the specified QSPI interrupt.

**Parameters:**

- `__HANDLE__`: specifies the QSPI Handle.
- `__INTERRUPT__`: specifies the QSPI interrupt source to disable. This parameter can be one of the following values:
  - `QSPI_IT_TO`: QSPI Timeout interrupt
  - `QSPI_IT_SM`: QSPI Status match interrupt
  - `QSPI_IT_FT`: QSPI FIFO threshold interrupt
  - `QSPI_IT_TC`: QSPI Transfer complete interrupt
  - `QSPI_IT_TE`: QSPI Transfer error interrupt

**Return value:**

- None

**Description:**

- Checks whether the specified QSPI interrupt source is enabled.

**Parameters:**

- `__HANDLE__`: specifies the QSPI Handle.
- `__INTERRUPT__`: specifies the QSPI interrupt source to check. This parameter can be one of the following values:
  - `QSPI_IT_TO`: QSPI Time out interrupt
  - `QSPI_IT_SM`: QSPI Status match

`__HAL_QSPI_DISABLE_IT``__HAL_QSPI_GET_IT_SOURCE`

- interrupt
- QSPI\_IT\_FT: QSPI FIFO threshold interrupt
- QSPI\_IT\_TC: QSPI Transfer complete interrupt
- QSPI\_IT\_TE: QSPI Transfer error interrupt

**Return value:**

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

**Description:**

- Get the selected QSPI's flag status.

**Parameters:**

- \_\_HANDLE\_\_: specifies the QSPI Handle.
- \_\_FLAG\_\_: specifies the QSPI flag to check. This parameter can be one of the following values:
  - QSPI\_FLAG\_BUSY: QSPI Busy flag
  - QSPI\_FLAG\_TO: QSPI Time out flag
  - QSPI\_FLAG\_SM: QSPI Status match flag
  - QSPI\_FLAG\_FT: QSPI FIFO threshold flag
  - QSPI\_FLAG\_TC: QSPI Transfer complete flag
  - QSPI\_FLAG\_TE: QSPI Transfer error flag

**Return value:**

- None

**Description:**

- Clears the specified QSPI's flag status.

**Parameters:**

- \_\_HANDLE\_\_: specifies the QSPI Handle.
- \_\_FLAG\_\_: specifies the QSPI clear register flag that needs to be set This parameter can be one of the following values:
  - QSPI\_FLAG\_TO: QSPI Time out flag
  - QSPI\_FLAG\_SM: QSPI Status match flag
  - QSPI\_FLAG\_TC: QSPI Transfer complete flag
  - QSPI\_FLAG\_TE: QSPI Transfer error flag

**Return value:**

- None

\_\_HAL\_QSPI\_GET\_FLAG

\_\_HAL\_QSPI\_CLEAR\_FLAG

**QSPI Fifo Threshold**

IS\_QSPI\_FIFO\_THRESHOLD

**QSPI Flags**

QSPI_FLAG_BUSY	Busy flag: operation is ongoing
QSPI_FLAG_TO	Timeout flag: timeout occurs in memory-mapped mode
QSPI_FLAG_SM	Status match flag: received data matches in autopolling mode
QSPI_FLAG_FT	Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete
QSPI_FLAG_TC	Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted
QSPI_FLAG_TE	Transfer error flag: invalid address is being accessed

**QSPI Flash Size**

IS\_QSPI\_FLASH\_SIZE

**QSPI Flash Select**

QSPI\_FLASH\_ID\_1

QSPI\_FLASH\_ID\_2

**QSPI Instruction**

IS\_QSPI\_INSTRUCTION

**QSPI Instruction Mode**

QSPI_INSTRUCTION_NONE	No instruction
QSPI_INSTRUCTION_1_LINE	Instruction on a single line
QSPI_INSTRUCTION_2_LINES	Instruction on two lines
QSPI_INSTRUCTION_4_LINES	Instruction on four lines

**QSPI Interrupts**

QSPI_IT_TO	Interrupt on the timeout flag
QSPI_IT_SM	Interrupt on the status match flag
QSPI_IT_FT	Interrupt on the fifo threshold flag
QSPI_IT_TC	Interrupt on the transfer complete flag
QSPI_IT_TE	Interrupt on the transfer error flag

**QSPI Interval**

IS\_QSPI\_INTERVAL

**QSPI Match Mode**

QSPI_MATCH_MODE_AND	AND match mode between unmasked bits
QSPI_MATCH_MODE_OR	OR match mode between unmasked bits

**QSPI Private Constants**

QSPI_FUNCTIONAL_MODE_INDIRECT_WRITE	Indirect write mode
QSPI_FUNCTIONAL_MODE_INDIRECT_READ	Indirect read mode

QSPI\_FUNCTIONAL\_MODE\_AUTO\_POLLING      Automatic polling mode

QSPI\_FUNCTIONAL\_MODE\_MEMORY\_MAPPED      Memory-mapped mode

### **QSPI Private Macros**

IS\_QSPI\_FUNCTIONAL\_MODE

IS\_QSPI\_SSHIFT

IS\_QSPI\_CS\_HIGH\_TIME

IS\_QSPI\_CLOCK\_MODE

IS\_QSPI\_FLASH\_ID

IS\_QSPI\_DUAL\_FLASH\_MODE

IS\_QSPI\_ADDRESS\_SIZE

IS\_QSPI\_ALTERNATE\_BYTES\_SIZE

IS\_QSPI\_INSTRUCTION\_MODE

IS\_QSPI\_ADDRESS\_MODE

IS\_QSPI\_ALTERNATE\_BYTES\_MODE

IS\_QSPI\_DATA\_MODE

IS\_QSPI\_DDR\_MODE

IS\_QSPI\_DDR\_HHC

IS\_QSPI\_SIOO\_MODE

IS\_QSPI\_MATCH\_MODE

IS\_QSPI\_AUTOMATIC\_STOP

IS\_QSPI\_TIMEOUT\_ACTIVATION

IS\_QSPI\_GET\_FLAG

IS\_QSPI\_IT

### **QSPI Sample Shifting**

QSPI\_SAMPLE\_SHIFTING\_NONE      No clock cycle shift to sample data

QSPI\_SAMPLE\_SHIFTING\_HALFCYCLE      1/2 clock cycle shift to sample data

### **QSPI SIOO Mode**

QSPI\_SIOO\_INST\_EVERY\_CMD      Send instruction on every transaction

QSPI\_SIOO\_INST\_ONLY\_FIRST\_CMD      Send instruction only for the first command

### **QSPI Status Bytes Size**

IS\_QSPI\_STATUS\_BYTES\_SIZE

### **QSPI TimeOut Activation**

QSPI\_TIMEOUT\_COUNTER\_DISABLE      Timeout counter disabled, nCS remains active

QSPI\_TIMEOUT\_COUNTER\_ENABLE      Timeout counter enabled, nCS released when timeout expires

### **QSPI TimeOut Period**

IS\_QSPI\_TIMEOUT\_PERIOD

***QSPI Timeout definition***`HAL_QPSI_TIMEOUT_DEFAULT_VALUE`



## 43 HAL RCC Generic Driver

### 43.1 RCC Firmware driver registers structures

#### 43.1.1 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 = 2 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, SDMMC and RNG clocks. This parameter must be a number between Min\_Data = 2 and Max\_Data = 15

#### 43.1.2 RCC\_OscInitTypeDef

##### Data Fields

- *uint32\_t OscillatorType*
- *uint32\_t HSEState*
- *uint32\_t LSEState*
- *uint32\_t HSIState*
- *uint32\_t HSCalibrationValue*
- *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 (default is RCC\_HSICALIBRATION\_DEFAULT). This parameter must be a number between Min\_Data = 0x00 and Max\_Data = 0x1F
- ***uint32\_t RCC\_OscInitTypeDef::LSIState***  
The new state of the LSI. This parameter can be a value of [RCC\\_LSI\\_Config](#)
- ***RCC\_PLLInitTypeDef RCC\_OscInitTypeDef::PLL***  
PLL structure parameters

**43.1.3 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](#)

## 43.2 RCC Firmware driver API description

### 43.2.1 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)

### 43.2.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
- If peripheral is mapped on AHB: the delay is 2 AHB clock cycle after the clock enable bit is set on the hardware register
- If peripheral is mapped on APB: the delay is 2 APB clock cycle after the clock enable bit is set on the hardware register

Implemented Workaround:

- For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each `__HAL_RCC_PPP_CLK_ENABLE()` macro.

### 43.2.3 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 buses 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 216 MHz)
  - The second output is used to generate the clock for the USB OTG FS (48 MHz), the random analog generator ( $\leq 48$  MHz) and the SDIO ( $\leq 48$  MHz).

6. CSS (Clock security system), once enable using the function `HAL_RCC_EnableCSS()` and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M7 NMI (Non-Maskable Interrupt) exception vector.
7. MCO1 (microcontroller clock output), used to output HSI, LSE, HSE or PLL clock (through a configurable prescaler) on PA8 pin.
8. MCO2 (microcontroller clock output), used to output HSE, PLL, SYSCLK or PLLI2S clock (through a configurable prescaler) on PC9 pin.

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE and PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "`HAL_RCC_GetSysClockFreq()`" function to retrieve the frequencies of these clocks. All the peripheral clocks are derived from the System clock (SYSCLK) except: I2S: the I2S clock can be derived either from a specific PLL (PLLI2S) or from an external clock mapped on the I2S\_CKIN pin. You have to use `__HAL_RCC_PLLI2S_CONFIG()` macro to configure this clock. SAI: the SAI clock can be derived either from a specific PLL (PLLI2S) or (PLLSAI) or from an external clock mapped on the I2S\_CKIN pin. You have to use `__HAL_RCC_PLLI2S_CONFIG()` macro to configure this clock. RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use `__HAL_RCC_RTC_CONFIG()` and `__HAL_RCC_RTC_ENABLE()` macros to configure this clock. USB OTG FS, SDIO and RTC: USB OTG FS require a frequency equal to 48 MHz to work correctly, while the SDIO require a frequency equal or lower than 48. This clock is derived of the main PLL through PLLQ divider. IWDG clock which is always the LSI clock.

This section contains the following APIs:

- [`HAL\_RCC\_DeInit\(\)`](#)
- [`HAL\_RCC\_OscConfig\(\)`](#)
- [`HAL\_RCC\_ClockConfig\(\)`](#)

#### 43.2.4 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\(\)`](#)

#### 43.2.5 HAL\_RCC\_DeInit

Function Name	<b>void HAL_RCC_DeInit (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 source HSE, PLL and PLLI2S OFF AHB, APB1 and APB2 prescaler set to 1. CSS, MCO1 and MCO2 OFF All interrupts disabled</li> <li>• This function doesn't modify the configuration of the Peripheral clocks LSI, LSE and RTC clocks</li> </ul>

### 43.2.6 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>

### 43.2.7 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_OscInitTypeDef 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. You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.</li> <li>• Depending on the device voltage range, the software has to</li> </ul>

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")

### 43.2.8 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 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 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</li> <li>• <b>RCC_MCODiv:</b> specifies the MCOx 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</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• PA8/PC9 should be configured in alternate function mode.</li> </ul>

### 43.2.9 HAL\_RCC\_EnableCSS

Function Name	<b>void HAL_RCC_EnableCSS (void )</b>
Function Description	Enables the Clock Security System.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• 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-M7 NMI (Non-Maskable Interrupt) exception vector.</li> </ul>

**43.2.10 HAL\_RCC\_DisableCSS**

Function Name	<b>void HAL_RCC_DisableCSS (void )</b>
Function Description	Disables the Clock Security System.
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**43.2.11 HAL\_RCC\_GetSysClockFreq**

Function Name	<b>uint32_t HAL_RCC_GetSysClockFreq (void )</b>
Function Description	Returns the SYSCLK frequency.
Return values	<ul style="list-style-type: none"> <li>• SYSCLK frequency</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:</li> <li>• If SYSCLK source is HSI, function returns values based on HSI_VALUE(*)</li> <li>• If SYSCLK source is HSE, function returns values based on HSE_VALUE(**)</li> <li>• If SYSCLK source is PLL, function returns values based on HSE_VALUE(**) or HSI_VALUE(*) multiplied/divided by the PLL factors.</li> <li>• (*) HSI_VALUE is a constant defined in stm32f7xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.</li> <li>• (**) HSE_VALUE is a constant defined in stm32f7xx_hal_conf.h file (default value 25 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.</li> <li>• The result of this function could be not correct when using fractional value for HSE crystal.</li> <li>• This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.</li> <li>• Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.</li> </ul>

**43.2.12 HAL\_RCC\_GetHCLKFreq**

Function Name	<b>uint32_t HAL_RCC_GetHCLKFreq (void )</b>
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>

**43.2.13 HAL\_RCC\_GetPCLK1Freq**



Function Name	<b>uint32_t HAL_RCC_GetPCLK1Freq (void )</b>
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>

#### 43.2.14 HAL\_RCC\_GetPCLK2Freq

Function Name	<b>uint32_t HAL_RCC_GetPCLK2Freq (void )</b>
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>

#### 43.2.15 HAL\_RCC\_GetOscConfig

Function Name	<b>void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)</b>
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li><b>RCC_OscInitStruct:</b> pointer to an RCC_OscInitTypeDef structure that will be configured.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 43.2.16 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>

#### 43.2.17 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>

#### 43.2.18 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>

## 43.3 RCC Firmware driver defines

### 43.3.1 RCC

#### ***AHB1 Peripheral Clock Enable Disable***

\_\_HAL\_RCC\_CRC\_CLK\_ENABLE  
 \_\_HAL\_RCC\_DMA1\_CLK\_ENABLE  
 \_\_HAL\_RCC\_CRC\_CLK\_DISABLE  
 \_\_HAL\_RCC\_DMA1\_CLK\_DISABLE

#### ***AHB1 Peripheral Clock Sleep Enable Disable Status***

\_\_HAL\_RCC\_CRC\_IS\_CLK\_SLEEP\_ENABLED  
 \_\_HAL\_RCC\_DMA1\_IS\_CLK\_SLEEP\_ENABLED  
 \_\_HAL\_RCC\_CRC\_IS\_CLK\_SLEEP\_DISABLED  
 \_\_HAL\_RCC\_DMA1\_IS\_CLK\_SLEEP\_DISABLED

#### ***AHB1 Peripheral Clock Enable Disable Status***

\_\_HAL\_RCC\_CRC\_IS\_CLK\_ENABLED  
 \_\_HAL\_RCC\_DMA1\_IS\_CLK\_ENABLED  
 \_\_HAL\_RCC\_CRC\_IS\_CLK\_DISABLED  
 \_\_HAL\_RCC\_DMA1\_IS\_CLK\_DISABLED

#### ***RCC AHB Clock Source***

RCC\_SYSCLK\_DIV1  
 RCC\_SYSCLK\_DIV2  
 RCC\_SYSCLK\_DIV4  
 RCC\_SYSCLK\_DIV8  
 RCC\_SYSCLK\_DIV16  
 RCC\_SYSCLK\_DIV64  
 RCC\_SYSCLK\_DIV128  
 RCC\_SYSCLK\_DIV256  
 RCC\_SYSCLK\_DIV512

#### ***RCC 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\_WWDG\_CLK\_ENABLE

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE

\_\_HAL\_RCC\_WWDG\_CLK\_DISABLE

\_\_HAL\_RCC\_PWR\_CLK\_DISABLE

**APB1 Peripheral Clock Enable Disable Status**

\_\_HAL\_RCC\_WWDG\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_PWR\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_WWDG\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_PWR\_IS\_CLK\_DISABLED

**APB1 Peripheral Clock Sleep Enable Disable Status**

\_\_HAL\_RCC\_WWDG\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_PWR\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_WWDG\_IS\_CLK\_SLEEP\_DISABLED

\_\_HAL\_RCC\_PWR\_IS\_CLK\_SLEEP\_DISABLED

**APB1 Force Release Reset**

\_\_HAL\_RCC\_APB1\_FORCE\_RESET

\_\_HAL\_RCC\_WWDG\_FORCE\_RESET

\_\_HAL\_RCC\_PWR\_FORCE\_RESET

\_\_HAL\_RCC\_APB1\_RELEASE\_RESET

\_\_HAL\_RCC\_WWDG\_RELEASE\_RESET

\_\_HAL\_RCC\_PWR\_RELEASE\_RESET

**APB2 Peripheral Clock Enable Disable**

\_\_HAL\_RCC\_SYSCFG\_CLK\_ENABLE

\_\_HAL\_RCC\_SYSCFG\_CLK\_DISABLE

**APB2 Peripheral Clock Enable Disable Status**

\_\_HAL\_RCC\_SYSCFG\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_SYSCFG\_IS\_CLK\_DISABLED

**APB2 Peripheral Clock Sleep Enable Disable Status**

\_\_HAL\_RCC\_SYSCFG\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_SYSCFG\_IS\_CLK\_SLEEP\_DISABLED

**APB2 Force Release Reset**

\_\_HAL\_RCC\_APB2\_FORCE\_RESET

\_\_HAL\_RCC\_SYSCFG\_FORCE\_RESET

\_\_HAL\_RCC\_APB2\_RELEASE\_RESET

\_\_HAL\_RCC\_SYSCFG\_RELEASE\_RESET

***RCC BitAddress Alias***

RCC\_CIR\_BYTE1\_ADDRESS

RCC\_CIR\_BYTE2\_ADDRESS

RCC\_DBP\_TIMEOUT\_VALUE

RCC\_LSE\_TIMEOUT\_VALUE

***AHB/APB Peripheral Clock Sleep Enable Disable Status***

\_\_HAL\_RCC\_FLITF\_IS\_CLK\_SLEEP\_ENABLED

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

\_\_HAL\_RCC\_AXI\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_SRAM1\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_SRAM2\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_BKPSRAM\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_DTCM\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_DMA2\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_DMA2D\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_ETHMAC\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_ETHMACTX\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_ETHMACRX\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_ETHMACPTP\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_USB\_OTG\_HS\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_USB\_OTG\_HS\_ULPI\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_GPIOA\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_GPIOB\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_GPIOC\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_GPIOD\_IS\_CLK\_SLEEP\_ENABLED

\_\_HAL\_RCC\_GPIOE\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_GPIOF\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_GPIOG\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_GPIOH\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_GPIOI\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_GPIOJ\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_GPIOK\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_FLITF\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_AXI\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SRAM1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SRAM2\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_BKPSRAM\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_DTCM\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_DMA2\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_DMA2D\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_ETHMAC\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_ETHMACTX\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_ETHMACRX\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_ETHMACPTP\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_USB\_OTG\_HS\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_USB\_OTG\_HS\_ULPI\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOA\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOB\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOC\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOD\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOE\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOF\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOG\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOH\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOI\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOJ\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_GPIOK\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_DCMI\_IS\_CLK\_SLEEP\_ENABLED

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power

consumption.  
After wakeup  
from SLEEP  
mode, the  
peripheral clock is  
enabled again. By  
default, all  
peripheral clocks  
are enabled  
during SLEEP  
mode.

```
__HAL_RCC_DCMI_IS_CLK_SLEEP_DISABLED  
__HAL_RCC_RNG_IS_CLK_SLEEP_ENABLED  
__HAL_RCC_RNG_IS_CLK_SLEEP_DISABLED  
__HAL_RCC_USB_OTG_FS_IS_CLK_SLEEP_ENABLED  
__HAL_RCC_USB_OTG_FS_IS_CLK_SLEEP_DISABLED  
__HAL_RCC_Cryp_IS_CLK_SLEEP_ENABLED  
__HAL_RCC_HASH_IS_CLK_SLEEP_ENABLED  
__HAL_RCC_Cryp_IS_CLK_SLEEP_DISABLED  
__HAL_RCC_HASH_IS_CLK_SLEEP_DISABLED  
__HAL_RCC_FMC_IS_CLK_SLEEP_ENABLED
```

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

```
__HAL_RCC_FMC_IS_CLK_SLEEP_DISABLED  
__HAL_RCC_QSPI_IS_CLK_SLEEP_ENABLED  
__HAL_RCC_QSPI_IS_CLK_SLEEP_DISABLED  
__HAL_RCC_TIM2_IS_CLK_SLEEP_ENABLED
```

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption.

After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

```
__HAL_RCC_TIM3_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM4_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM5_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM6_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM7_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM12_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM13_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM14_IS_CLK_SLEEP_ENABLED
__HAL_RCC_LPTIM1_IS_CLK_SLEEP_ENABLED
__HAL_RCC_SPI2_IS_CLK_SLEEP_ENABLED
__HAL_RCC_SPI3_IS_CLK_SLEEP_ENABLED
__HAL_RCC_SPDIFRX_IS_CLK_SLEEP_ENABLED
__HAL_RCC_USART2_IS_CLK_SLEEP_ENABLED
__HAL_RCC_USART3_IS_CLK_SLEEP_ENABLED
__HAL_RCC_UART4_IS_CLK_SLEEP_ENABLED
__HAL_RCC_UART5_IS_CLK_SLEEP_ENABLED
__HAL_RCC_I2C1_IS_CLK_SLEEP_ENABLED
__HAL_RCC_I2C2_IS_CLK_SLEEP_ENABLED
__HAL_RCC_I2C3_IS_CLK_SLEEP_ENABLED
__HAL_RCC_I2C4_IS_CLK_SLEEP_ENABLED
__HAL_RCC_CAN1_IS_CLK_SLEEP_ENABLED
__HAL_RCC_CAN2_IS_CLK_SLEEP_ENABLED
__HAL_RCC_CEC_IS_CLK_SLEEP_ENABLED
__HAL_RCC_DAC_IS_CLK_SLEEP_ENABLED
__HAL_RCC_UART7_IS_CLK_SLEEP_ENABLED
__HAL_RCC_UART8_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM2_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM3_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM4_IS_CLK_SLEEP_DISABLED
```

\_\_HAL\_RCC\_TIM5\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM6\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM7\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM12\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM13\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM14\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_LPTIM1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SPI2\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SPI3\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SPDIFRX\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_USART2\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_USART3\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_UART4\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_UART5\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_I2C1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_I2C2\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_I2C3\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_I2C4\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_CAN1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_CAN2\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_CEC\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_DAC\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_UART7\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_UART8\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM1\_IS\_CLK\_SLEEP\_ENABLED

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

\_\_HAL\_RCC\_TIM8\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_USART1\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_USART6\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_ADC1\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_ADC2\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_ADC3\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_SDMMC1\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_SPI1\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_SPI4\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_TIM9\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_TIM10\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_TIM11\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_SPI5\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_SPI6\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_SAI1\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_SAI2\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_LTDC\_IS\_CLK\_SLEEP\_ENABLED  
\_\_HAL\_RCC\_TIM1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM8\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_USART1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_USART6\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_ADC1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_ADC2\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_ADC3\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SDMMC1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SPI1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SPI4\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM9\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM10\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_TIM11\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SPI5\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SPI6\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SAI1\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_SAI2\_IS\_CLK\_SLEEP\_DISABLED  
\_\_HAL\_RCC\_LTDC\_IS\_CLK\_SLEEP\_DISABLED

**RCC Flags**



RCC\_FLAG\_HSIRDY  
RCC\_FLAG\_HSERDY  
RCC\_FLAG\_PLLRDY  
RCC\_FLAG\_PLI2SRDY  
RCC\_FLAG\_PLLSAIRDY  
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:**

- `__INTERERRUPT__`: 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_PLI2SRDY`: 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:**

- `__INTERERRUPT__`: 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:**

- `__INTERERRUPT__`: 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:**

- `__INTERERRUPT__`: 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 `__INTERERRUPT__` (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_HSIIRDY`: 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`

**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_LSEDRIVE_CONFIG`

**Description:**

- Macro to configures the External Low Speed oscillator (LSE) drive capability.

**Parameters:**

- `__RCC_LSEDRIVE__`: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
  - `RCC_LSEDRIVE_LOW`: LSE oscillator low drive capability.
  - `RCC_LSEDRIVE_MEDIUMLOW`: LSE oscillator medium low drive capability.
  - `RCC_LSEDRIVE_MEDIUMHIGH`: LSE oscillator medium high drive capability.
  - `RCC_LSEDRIVE_HIGH`: LSE oscillator high drive capability.

**Return value:**

- None

**Notes:**

- As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using `HAL_PWR_EnableBkUpAccess()` function before to configure the LSE (to be done once after reset).

`__HAL_RCC_GET_PLL_OSCSOURCE`

**Description:**

- Macro to get the oscillator used as PLL clock source.

**Return value:**

- The: oscillator used as PLL clock source. The returned value can be one of the following:
  - `RCC_PLLSOURCE_HSI`: HSI oscillator is used as PLL clock source.
  - `RCC_PLLSOURCE_HSE`: HSE oscillator is

used as PLL clock source.

### **RCC HSE Config**

RCC\_HSE\_OFF

RCC\_HSE\_ON

RCC\_HSE\_BYPASS

### **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:**

- 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.

### **RCC 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, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or

indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. 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.

**RTC Clock Configuration**

\_\_HAL\_RCC\_RTC\_ENABLE

**Notes:**

- These macros must be used only after the RTC clock source was selected.

\_\_HAL\_RCC\_RTC\_DISABLE

\_\_HAL\_RCC\_RTC\_CLKPRESCALER

**Description:**

- Macros to configure the RTC clock (RTCCLK).

**Parameters:**

- \_\_RTCCLKSource\_\_: specifies the RTC clock source. This parameter can be one of the following values:
  - RCC\_RTCCLKSOURCE\_LSE: LSE selected as RTC clock.

- RCC\_RTCCLKSOURCE\_LSI: LSI selected as RTC clock.
- RCC\_RTCCLKSOURCE\_HSE\_DIVx: HSE clock divided by x selected as RTC clock, where x:[2,31]

**Notes:**

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it can't be changed unless the Backup domain is reset using `__HAL_RCC_BackupReset_RELEASE()` macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

`__HAL_RCC_RTC_CONFIG``__HAL_RCC_BACKUPRESET_FORCE`**Notes:**

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC\_CSR register. The BKPSRAM is not affected by this reset.

`__HAL_RCC_BACKUPRESET_RELEASE`**RCC Interrupt**`RCC_IT_LSIRDY``RCC_IT_LSERDY``RCC_IT_HSIRDY``RCC_IT_HSERDY``RCC_IT_PLLRDY``RCC_IT_PLLI2SRDY``RCC_IT_PLLSAIRDY``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  
IS\_RCC\_RTCCLKSOURCE  
IS\_RCC\_LSE\_DRIVE

***RCC LSE Drive configurations***

RCC\_LSEDRIVE\_LOW  
RCC\_LSEDRIVE\_MEDIUMLOW  
RCC\_LSEDRIVE\_MEDIUMHIGH  
RCC\_LSEDRIVE\_HIGH

***RCC LSE Config***

RCC\_LSE\_OFF  
RCC\_LSE\_ON  
RCC\_LSE\_BYPASS

***LSE Configuration***

\_\_HAL\_RCC\_LSE\_CONFIG

**Description:**

- Macro to configure the External Low Speed oscillator (LSE).

**Parameters:**

- `__STATE__`: specifies the new state of the LSE. This parameter can be one of the following values:
  - `RCC_LSE_OFF`: turn OFF the LSE oscillator,



LSERDY flag goes low after 6 LSE oscillator clock cycles.

- RCC\_LSE\_ON: turn ON the LSE oscillator.
- RCC\_LSE\_BYPASS: LSE oscillator bypassed with external clock.

**Notes:**

- Transition LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL\_PWR\_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC\_LSE\_ON or RCC\_LSE\_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

**RCC LSI Config**

RCC\_LSI\_OFF

RCC\_LSI\_ON

**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

**RCC MCO1 Clock Source**

RCC\_MCO1SOURCE\_HSI

RCC\_MCO1SOURCE\_LSE

RCC\_MCO1SOURCE\_HSE

RCC\_MCO1SOURCE\_PLLCLK

**RCC MCO2 Clock Source**

RCC\_MCO2SOURCE\_SYSCLK

RCC\_MCO2SOURCE\_PLLI2SCLK

RCC\_MCO2SOURCE\_HSE

RCC\_MCO2SOURCE\_PLLCLK

**RCC MCO1 Clock Prescaler**

RCC\_MCODIV\_1

RCC\_MCODIV\_2

RCC\_MCODIV\_3

RCC\_MCODIV\_4

RCC\_MCODIV\_5

***RCC MCO Index***

RCC\_MCO1

RCC\_MCO2

***Oscillator Type***

RCC\_OSCILLATORTYPE\_NONE

RCC\_OSCILLATORTYPE\_HSE

RCC\_OSCILLATORTYPE\_HSI

RCC\_OSCILLATORTYPE\_LSE

RCC\_OSCILLATORTYPE\_LSI

***RCC Peripheral Clock Force Release***

\_\_HAL\_RCC\_AHB1\_FORCE\_RESET

\_\_HAL\_RCC\_CRC\_FORCE\_RESET

\_\_HAL\_RCC\_DMA1\_FORCE\_RESET

\_\_HAL\_RCC\_AHB1\_RELEASE\_RESET

\_\_HAL\_RCC\_CRC\_RELEASE\_RESET

\_\_HAL\_RCC\_DMA1\_RELEASE\_RESET

***RCC Peripheral Clock Sleep Enable Disable***

\_\_HAL\_RCC\_CRC\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_DMA1\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_CRC\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_DMA1\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_WWDG\_CLK\_SLEEP\_ENABLE

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

\_\_HAL\_RCC\_PWR\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_WWDG\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_PWR\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_SYSCFG\_CLK\_SLEEP\_ENABLE

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the

peripheral clock is enabled again.  
By default, all peripheral clocks are enabled during SLEEP mode.

`__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE`

#### **PLL Clock Divider**

`RCC_PLLP_DIV2`

`RCC_PLLP_DIV4`

`RCC_PLLP_DIV6`

`RCC_PLLP_DIV8`

#### **PLL Clock Source**

`RCC_PLLSOURCE_HSI`

`RCC_PLLSOURCE_HSE`

#### **RCC 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, SDMMC 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.
- You have to set the PLLP parameter correctly to not exceed 216 MHz on the System clock frequency.
- 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 SDMMC and RNG need a frequency lower than or equal to 48 MHz to work correctly.

**\_\_HAL\_RCC\_PLL\_PLLSOURCE\_CONFIG****Description:**

- Macro to configure the PLL clock source.

**Parameters:**

- **\_\_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

\_\_HAL\_RCC\_PLL\_PLLM\_CONFIG

**Notes:**

- This function must be used only when the main PLL is disabled.

**Description:**

- Macro to configure the PLL multiplication factor.

**Parameters:**

- \_\_PLLM\_\_: specifies the division factor for PLL VCO input clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 63.

**Notes:**

- This function must be used only when the main PLL is disabled.
- 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.

**PLL I2S Configuration**

\_\_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.

\_\_HAL\_RCC\_PLLI2S\_ENABLE

**Notes:**

- The PLLI2S is disabled by hardware when entering STOP and STANDBY modes.

\_\_HAL\_RCC\_PLLI2S\_DISABLE

**RCC Private Constants**

HSE\_TIMEOUT\_VALUE

HSI\_TIMEOUT\_VALUE

LSI\_TIMEOUT\_VALUE

PLL\_TIMEOUT\_VALUE

CLOCKSWITCH\_TIMEOUT\_VALUE

***RCC Private Macros***

MCO1\_CLK\_ENABLE

MCO1\_GPIO\_PORT

MCO1\_PIN

MCO2\_CLK\_ENABLE

MCO2\_GPIO\_PORT

MCO2\_PIN

***RCC RTC Clock Source***

RCC\_RTCCLKSOURCE\_LSE

RCC\_RTCCLKSOURCE\_LSI

RCC\_RTCCLKSOURCE\_HSE\_DIV2

RCC\_RTCCLKSOURCE\_HSE\_DIV3

RCC\_RTCCLKSOURCE\_HSE\_DIV4

RCC\_RTCCLKSOURCE\_HSE\_DIV5

RCC\_RTCCLKSOURCE\_HSE\_DIV6

RCC\_RTCCLKSOURCE\_HSE\_DIV7

RCC\_RTCCLKSOURCE\_HSE\_DIV8

RCC\_RTCCLKSOURCE\_HSE\_DIV9

RCC\_RTCCLKSOURCE\_HSE\_DIV10

RCC\_RTCCLKSOURCE\_HSE\_DIV11

RCC\_RTCCLKSOURCE\_HSE\_DIV12

RCC\_RTCCLKSOURCE\_HSE\_DIV13

RCC\_RTCCLKSOURCE\_HSE\_DIV14

RCC\_RTCCLKSOURCE\_HSE\_DIV15

RCC\_RTCCLKSOURCE\_HSE\_DIV16

RCC\_RTCCLKSOURCE\_HSE\_DIV17

RCC\_RTCCLKSOURCE\_HSE\_DIV18

RCC\_RTCCLKSOURCE\_HSE\_DIV19

RCC\_RTCCLKSOURCE\_HSE\_DIV20

RCC\_RTCCLKSOURCE\_HSE\_DIV21

RCC\_RTCCLKSOURCE\_HSE\_DIV22

RCC\_RTCCLKSOURCE\_HSE\_DIV23

RCC\_RTCCLKSOURCE\_HSE\_DIV24

RCC\_RTCCLKSOURCE\_HSE\_DIV25

RCC\_RTCCLKSOURCE\_HSE\_DIV26

RCC\_RTCCLKSOURCE\_HSE\_DIV27

RCC\_RTCCLKSOURCE\_HSE\_DIV28

RCC\_RTCCLKSOURCE\_HSE\_DIV29

RCC\_RTCCLKSOURCE\_HSE\_DIV30

RCC\_RTCCLKSOURCE\_HSE\_DIV31

***RCC 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

***RCC System Clock Type***

RCC\_CLOCKTYPE\_SYSCLK

RCC\_CLOCKTYPE\_HCLK

RCC\_CLOCKTYPE\_PCLK1

RCC\_CLOCKTYPE\_PCLK2

## 44 HAL RCC Extension Driver

### 44.1 RCCEX Firmware driver registers structures

#### 44.1.1 RCC\_PLLI2SInitTypeDef

##### Data Fields

- *uint32\_t PLLI2SN*
- *uint32\_t PLLI2SR*
- *uint32\_t PLLI2SQ*
- *uint32\_t PLLI2SP*

##### 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 = 49 and Max\_Data = 432. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI
- ***uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SR***  
Specifies the division factor for I2S clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 7. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI
- ***uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SQ***  
Specifies the division factor for SAI1 clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 15. This parameter will be used only when PLLI2S is selected as Clock Source SAI
- ***uint32\_t RCC\_PLLI2SInitTypeDef::PLLI2SP***  
Specifies the division factor for SPDIF-RX clock. This parameter must be a number between 0 and 3 for respective values 2, 4, 6 and 8. This parameter will be used only when PLLI2S is selected as Clock Source SPDDIF-RX

#### 44.1.2 RCC\_PLLSAIInitTypeDef

##### Data Fields

- *uint32\_t PLLSAIN*
- *uint32\_t PLLSAIQ*
- *uint32\_t PLLSAIR*
- *uint32\_t PLLSAIP*

##### Field Documentation

- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIN***  
Specifies the multiplication factor for PLLI2S VCO output clock. This parameter must be a number between Min\_Data = 49 and Max\_Data = 432. This parameter will be used only when PLLSAI is selected as Clock Source SAI or LTDC



- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIQ***  
Specifies the division factor for SAI1 clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 15. This parameter will be used only when PLLSAI is selected as Clock Source SAI or LTDC
- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIR***  
specifies the division factor for LTDC clock This parameter must be a number between Min\_Data = 2 and Max\_Data = 7. This parameter will be used only when PLLSAI is selected as Clock Source LTDC
- ***uint32\_t RCC\_PLLSAIInitTypeDef::PLLSAIP***  
Specifies the division factor for 48MHz clock. This parameter can be a value of [RCCEX\\_PLLSAIP\\_Clock\\_Divider](#) This parameter will be used only when PLLSAI is disabled

### 44.1.3 RCC\_PeriphCLKInitTypeDef

#### Data Fields

- ***uint32\_t PeriphClockSelection***
- ***RCC\_PLLI2SInitTypeDef PLLI2S***
- ***RCC\_PLLSAIInitTypeDef PLLSAI***
- ***uint32\_t PLLI2SDivQ***
- ***uint32\_t PLLSAIDivQ***
- ***uint32\_t PLLSAIDivR***
- ***uint32\_t RTCClockSelection***
- ***uint32\_t I2sClockSelection***
- ***uint32\_t TIMPresSelection***
- ***uint32\_t Sai1ClockSelection***
- ***uint32\_t Sai2ClockSelection***
- ***uint32\_t Usart1ClockSelection***
- ***uint32\_t Usart2ClockSelection***
- ***uint32\_t Usart3ClockSelection***
- ***uint32\_t Uart4ClockSelection***
- ***uint32\_t Uart5ClockSelection***
- ***uint32\_t Usart6ClockSelection***
- ***uint32\_t Uart7ClockSelection***
- ***uint32\_t Uart8ClockSelection***
- ***uint32\_t I2c1ClockSelection***
- ***uint32\_t I2c2ClockSelection***
- ***uint32\_t I2c3ClockSelection***
- ***uint32\_t I2c4ClockSelection***
- ***uint32\_t Lptim1ClockSelection***
- ***uint32\_t CecClockSelection***
- ***uint32\_t Clk48ClockSelection***
- ***uint32\_t Sdmmc1ClockSelection***

#### Field Documentation

- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PeriphClockSelection***  
The Extended Clock to be configured. This parameter can be a value of [RCCEX\\_Periph\\_Clock\\_Selection](#)

- ***RCC\_PLLI2SInitTypeDef RCC\_PeriphCLKInitTypeDef::PLLI2S***  
PLL I2S structure parameters. This parameter will be used only when PLLI2S is selected as Clock Source I2S or SAI
- ***RCC\_PLLSAIInitTypeDef RCC\_PeriphCLKInitTypeDef::PLLSAI***  
PLL SAI structure parameters. This parameter will be used only when PLLI2S is selected as Clock Source SAI or LTDC
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PLLI2SDivQ***  
Specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between Min\_Data = 1 and Max\_Data = 32 This parameter will be used only when PLLI2S is selected as Clock Source SAI
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PLLSAIDivQ***  
Specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between Min\_Data = 1 and Max\_Data = 32 This parameter will be used only when PLLSAI is selected as Clock Source SAI
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::PLLSAIDivR***  
Specifies the PLLSAI division factor for LTDC clock. This parameter must be one value of [RCCEX\\_PLLSAI\\_DIVR](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::RTCClockSelection***  
Specifies RTC Clock source Selection. This parameter can be a value of [RCC\\_RTC\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::I2sClockSelection***  
Specifies I2S Clock source Selection. This parameter can be a value of [RCCEX\\_I2S\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::TIMPresSelection***  
Specifies TIM Clock Prescalers Selection. This parameter can be a value of [RCCEX\\_TIM\\_Prescaler\\_Selection](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Sai1ClockSelection***  
Specifies SAI1 Clock Prescalers Selection This parameter can be a value of [RCCEX\\_SAI1\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Sai2ClockSelection***  
Specifies SAI2 Clock Prescalers Selection This parameter can be a value of [RCCEX\\_SAI2\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Usart1ClockSelection***  
USART1 clock source This parameter can be a value of [RCCEX\\_USART1\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Usart2ClockSelection***  
USART2 clock source This parameter can be a value of [RCCEX\\_USART2\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Usart3ClockSelection***  
USART3 clock source This parameter can be a value of [RCCEX\\_USART3\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Uart4ClockSelection***  
UART4 clock source This parameter can be a value of [RCCEX\\_UART4\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Uart5ClockSelection***  
UART5 clock source This parameter can be a value of [RCCEX\\_UART5\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Usart6ClockSelection***  
USART6 clock source This parameter can be a value of [RCCEX\\_USART6\\_Clock\\_Source](#)
- ***uint32\_t RCC\_PeriphCLKInitTypeDef::Uart7ClockSelection***  
UART7 clock source This parameter can be a value of [RCCEX\\_UART7\\_Clock\\_Source](#)

- **`uint32_t RCC_PeriphCLKInitTypeDef::Uart8ClockSelection`**  
UART8 clock source This parameter can be a value of [RCCEx\\_UART8\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection`**  
I2C1 clock source This parameter can be a value of [RCCEx\\_I2C1\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c2ClockSelection`**  
I2C2 clock source This parameter can be a value of [RCCEx\\_I2C2\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection`**  
I2C3 clock source This parameter can be a value of [RCCEx\\_I2C3\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection`**  
I2C4 clock source This parameter can be a value of [RCCEx\\_I2C4\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection`**  
Specifies LPTIM1 clock source This parameter can be a value of [RCCEx\\_LPTIM1\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::CecClockSelection`**  
CEC clock source This parameter can be a value of [RCCEx\\_CEC\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::Clk48ClockSelection`**  
Specifies 48Mhz clock source used by USB OTG FS, RNG and SDMMC This parameter can be a value of [RCCEx\\_CLK48\\_Clock\\_Source](#)
- **`uint32_t RCC_PeriphCLKInitTypeDef::Sdmmc1ClockSelection`**  
SDMMC1 clock source This parameter can be a value of [RCCEx\\_SDMMC1\\_Clock\\_Source](#)

## 44.2 RCCEx Firmware driver API description

### 44.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.



Important note: Care must be taken when `HAL_RCCEx_PeriphCLKConfig()` is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and `RCC_BDCR` register will be set to their reset values.

This section contains the following APIs:

- [HAL\\_RCCEx\\_PeriphCLKConfig\(\)](#)
- [HAL\\_RCCEx\\_GetPeriphCLKConfig\(\)](#)
- [HAL\\_RCCEx\\_GetPeriphCLKFreq\(\)](#)

### 44.2.2 HAL\_RCCEx\_PeriphCLKConfig

Function Name	<b><code>HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef * PeriphClkInit)</code></b>
Function Description	Initializes the RCC extended peripherals clocks according to the specified parameters in the <code>RCC_PeriphCLKInitTypeDef</code> .
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit:</b> pointer to an <code>RCC_PeriphCLKInitTypeDef</code> structure that contains the configuration information for the Extended Peripherals clocks(I2S, SAI, LTDC RTC, TIM, UARTs, USARTs, LTPIM, SDMMC...).</li> </ul>

Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Care must be taken when HAL_RCCEX_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.</li> </ul>

### 44.2.3 HAL\_RCCEX\_GetPeriphCLKConfig

Function Name	<b>void HAL_RCCEX_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)</b>
Function Description	Get the RCC_PeriphCLKInitTypeDef according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClkInit:</b> pointer to the configured RCC_PeriphCLKInitTypeDef structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 44.2.4 HAL\_RCCEX\_GetPeriphCLKFreq

Function Name	<b>uint32_t HAL_RCCEX_GetPeriphCLKFreq (uint32_t PeriphClk)</b>
Function Description	Return the peripheral clock frequency for a given peripheral(SAI..)
Parameters	<ul style="list-style-type: none"> <li>• <b>PeriphClk:</b> Peripheral clock identifier This parameter can be one of the following values: RCC_PERIPHCLK_SAI1: SAI1 peripheral clock RCC_PERIPHCLK_SAI2: SAI2 peripheral clock</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Frequency in KHz</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• Return 0 if peripheral clock identifier not managed by this API</li> </ul>

## 44.3 RCCEX Firmware driver defines

### 44.3.1 RCCEX

#### *RCCEX CEC Clock Source*

RCC\_CECCLKSOURCE\_LSE

RCC\_CECCLKSOURCE\_HSI

#### *RCCEX CLK48 Clock Source*

RCC\_CLK48SOURCE\_PLL

RCC\_CLK48SOURCE\_PLLSAIP

#### *RCCEX Exported Macros*

**\_\_HAL\_RCC\_TIMCLKPRESCALER**

#### **Description:**

- Macro to configure the Timers clocks prescalers.

#### **Parameters:**

- **\_\_PRESC\_\_**: specifies the Timers

clocks prescalers selection This parameter can be one of the following values:

- **RCC\_TIMPRES\_DESACTIVATED:** The Timers kernels clocks prescaler is equal to HPRE if PPREx is corresponding to division by 1 or 2, else it is equal to  $[(HPRE * PPREx) / 2]$  if PPREx is corresponding to division by 4 or more.
- **RCC\_TIMPRES\_ACTIVATED:** The Timers kernels clocks prescaler is equal to HPRE if PPREx is corresponding to division by 1, 2 or 4, else it is equal to  $[(HPRE * PPREx) / 4]$  if PPREx is corresponding to division by 8 or more.

`__HAL_RCC_PLLSAI_ENABLE`

#### Notes:

- The PLLSAI is disabled by hardware when entering STOP and STANDBY modes.

`__HAL_RCC_PLLSAI_DISABLE`

`__HAL_RCC_PLLSAI_CONFIG`

#### Description:

- Macro to configure the PLLSAI clock multiplication and division factors.

#### Parameters:

- `__PLLSAIN__`: specifies the multiplication factor for PLLSAI VCO output clock. This parameter must be a number between Min\_Data = 49 and Max\_Data = 432.
- `__PLLSAIQ__`: specifies the division factor for SAI clock This parameter must be a number between Min\_Data = 2 and Max\_Data = 15.
- `__PLLSAIR__`: specifies the division factor for LTDC clock This parameter must be a number between Min\_Data = 2 and Max\_Data = 7.
- `__PLLSAIP__`: specifies the division factor for USB, RNG, SDMMC clocks This parameter can be a value of

#### Notes:

- This function must be used only when the PLLSAI is disabled. PLLSAI clock source is common with the main PLL (configured in `RCC_PLLConfig` function )
- You have to set the PLLSAIN parameter correctly to ensure that the VCO output

`__HAL_RCC_PLLI2S_CONFIG`

frequency is between Min\_Data = 49 and Max\_Data = 432 MHz.

**Description:**

- Macro used by the SAI HAL driver 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.
- `__PLLI2SQ__`: specifies the division factor for SAI clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 15.
- `__PLLI2SR__`: specifies the division factor for I2S clock. This parameter must be a number between Min\_Data = 2 and Max\_Data = 7.
- `__PLLI2SP__`: specifies the division factor for SPDDIF-RX clock. This parameter can be a number between 0 and 3 for respective values 2, 4, 6 and 8

**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_PLLI2S_PLLSAICLKDIVQ_CONFIG`**Description:**

- Macro to configure the SAI clock Divider coming from PLLI2S.

**Parameters:**

- `__PLLI2SDivQ__`: specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between 1 and 32. SAI1 clock frequency =  $f(\text{PLLI2SQ}) / \text{__PLLI2SDivQ__}$

**Notes:**

- This function must be called before enabling the PLLI2S.

**\_\_HAL\_RCC\_PLLSAI\_PLLSAICLKDIVQ\_CONFIG****Description:**

- Macro to configure the SAI clock Divider coming from PLLSAI.

**Parameters:**

- \_\_PLLSAIDivQ\_\_**: specifies the PLLSAI division factor for SAI1 clock . This parameter must be a number between Min\_Data = 1 and Max\_Data = 32. SAI1 clock frequency =  $f(\text{PLLSAIQ}) / \text{__PLLSAIDivQ__}$

**Notes:**

- This function must be called before enabling the PLLSAI.

**\_\_HAL\_RCC\_PLLSAI\_PLLSAICLKDIVR\_CONFIG****Description:**

- Macro to configure the LTDC clock Divider coming from PLLSAI.

**Parameters:**

- \_\_PLLSAIDivR\_\_**: specifies the PLLSAI division factor for LTDC clock . This parameter must be a number between Min\_Data = 2 and Max\_Data = 16. LTDC clock frequency =  $f(\text{PLLSAIR}) / \text{__PLLSAIDivR__}$

**Notes:**

- This function must be called before enabling the PLLSAI.

**\_\_HAL\_RCC\_SAI1\_CONFIG****Description:**

- Macro to configure SAI1 clock source selection.

**Parameters:**

- \_\_SOURCE\_\_**: specifies the SAI1 clock source. This parameter can be one of the following values:
  - RCC\_SAI1CLKSOURCE\_PLLI2S**: PLLI2S\_Q clock divided by PLLI2SDIVQ used as SAI1 clock.
  - RCC\_SAI1CLKSOURCE\_PLLSAI**: PLLISAI\_Q clock divided by PLLSAIDIVQ used as SAI1 clock.
  - RCC\_SAI1CLKSOURCE\_PIN**: External clock mapped on the I2S\_CKIN pin used as SAI1 clock.

**Notes:**

- This function must be called before enabling PLLSAI, PLLI2S and the SAI

**\_\_HAL\_RCC\_GET\_SAI1\_SOURCE**

clock.

**Description:**

- Macro to get the SAI1 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_SAI1CLKSOURCE\_PLLI2S: PLLI2S\_Q clock divided by PLLI2SDIVQ used as SAI1 clock.
  - RCC\_SAI1CLKSOURCE\_PLLSAI: PLLISAI\_Q clock divided by PLLSAIDIVQ used as SAI1 clock.
  - RCC\_SAI1CLKSOURCE\_PIN: External clock mapped on the I2S\_CKIN pin used as SAI1 clock.

**\_\_HAL\_RCC\_SAI2\_CONFIG****Description:**

- Macro to configure SAI2 clock source selection.

**Parameters:**

- **\_\_SOURCE\_\_**: specifies the SAI2 clock source. This parameter can be one of the following values:
  - RCC\_SAI2CLKSOURCE\_PLLI2S: PLLI2S\_Q clock divided by PLLI2SDIVQ used as SAI2 clock.
  - RCC\_SAI2CLKSOURCE\_PLLSAI: PLLISAI\_Q clock divided by PLLSAIDIVQ used as SAI2 clock.
  - RCC\_SAI2CLKSOURCE\_PIN: External clock mapped on the I2S\_CKIN pin used as SAI2 clock.

**Notes:**

- This function must be called before enabling PLLSAI, PLLI2S and the SAI clock.

**\_\_HAL\_RCC\_GET\_SAI2\_SOURCE****Description:**

- Macro to get the SAI2 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_SAI2CLKSOURCE\_PLLI2S: PLLI2S\_Q clock divided by PLLI2SDIVQ used as SAI2 clock.
  - RCC\_SAI2CLKSOURCE\_PLLSAI: PLLISAI\_Q clock divided by PLLSAIDIVQ used as SAI2 clock.
  - RCC\_SAI2CLKSOURCE\_PIN:



External clock mapped on the I2S\_CKIN pin used as SAI2 clock.

\_\_HAL\_RCC\_PLLSAI\_ENABLE\_IT  
\_\_HAL\_RCC\_PLLSAI\_DISABLE\_IT  
\_\_HAL\_RCC\_PLLSAI\_CLEAR\_IT  
\_\_HAL\_RCC\_PLLSAI\_GET\_IT

**Description:**

- Check the PLLSAI RDY interrupt has occurred or not.

**Return value:**

- The: new state (TRUE or FALSE).

\_\_HAL\_RCC\_PLLSAI\_GET\_FLAG

**Description:**

- Check PLLSAI RDY flag is set or not.

**Return value:**

- The: new state (TRUE or FALSE).

\_\_HAL\_RCC\_GET\_I2SCLKSOURCE

**Description:**

- Macro to Get I2S clock source selection.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_I2SCLKSOURCE\_PLLI2S: PLLI2S VCO output clock divided by PLLI2SR used as I2S clock.
  - RCC\_I2SCLKSOURCE\_EXT: External clock mapped on the I2S\_CKIN pin used as I2S clock source

\_\_HAL\_RCC\_I2C1\_CONFIG

**Description:**

- Macro to configure the I2C1 clock (I2C1CLK).

**Parameters:**

- \_\_I2C1\_CLKSOURCE\_\_: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC\_I2C1CLKSOURCE\_PCLK1: PCLK1 selected as I2C1 clock
  - RCC\_I2C1CLKSOURCE\_HSI: HSI selected as I2C1 clock
  - RCC\_I2C1CLKSOURCE\_SYSCLK: System Clock selected as I2C1 clock

\_\_HAL\_RCC\_GET\_I2C1\_SOURCE

**Description:**

- Macro to get the I2C1 clock source.

**Return value:**

### \_\_HAL\_RCC\_I2C2\_CONFIG

- The: clock source can be one of the following values:
  - RCC\_I2C1CLKSOURCE\_PCLK1: PCLK1 selected as I2C1 clock
  - RCC\_I2C1CLKSOURCE\_HSI: HSI selected as I2C1 clock
  - RCC\_I2C1CLKSOURCE\_SYSCLK: System Clock selected as I2C1 clock

#### Description:

- Macro to configure the I2C2 clock (I2C2CLK).

#### Parameters:

- \_\_I2C2\_CLKSOURCE\_\_: specifies the I2C2 clock source. This parameter can be one of the following values:
  - RCC\_I2C2CLKSOURCE\_PCLK1: PCLK1 selected as I2C2 clock
  - RCC\_I2C2CLKSOURCE\_HSI: HSI selected as I2C2 clock
  - RCC\_I2C2CLKSOURCE\_SYSCLK: System Clock selected as I2C2 clock

### \_\_HAL\_RCC\_GET\_I2C2\_SOURCE

#### Description:

- Macro to get the I2C2 clock source.

#### Return value:

- The: clock source can be one of the following values:
  - RCC\_I2C2CLKSOURCE\_PCLK1: PCLK1 selected as I2C2 clock
  - RCC\_I2C2CLKSOURCE\_HSI: HSI selected as I2C2 clock
  - RCC\_I2C2CLKSOURCE\_SYSCLK: System Clock selected as I2C2 clock

### \_\_HAL\_RCC\_I2C3\_CONFIG

#### Description:

- Macro to configure the I2C3 clock (I2C3CLK).

#### Parameters:

- \_\_I2C3\_CLKSOURCE\_\_: specifies the I2C3 clock source. This parameter can be one of the following values:
  - RCC\_I2C3CLKSOURCE\_PCLK1: PCLK1 selected as I2C3 clock
  - RCC\_I2C3CLKSOURCE\_HSI: HSI selected as I2C3 clock
  - RCC\_I2C3CLKSOURCE\_SYSCLK: System Clock selected as I2C3

clock

**\_\_HAL\_RCC\_GET\_I2C3\_SOURCE****Description:**

- macro to get the I2C3 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_I2C3CLKSOURCE\_PCLK1: PCLK1 selected as I2C3 clock
  - RCC\_I2C3CLKSOURCE\_HSI: HSI selected as I2C3 clock
  - RCC\_I2C3CLKSOURCE\_SYSCLK: System Clock selected as I2C3 clock

**\_\_HAL\_RCC\_I2C4\_CONFIG****Description:**

- Macro to configure the I2C4 clock (I2C4CLK).

**Parameters:**

- \_\_I2C4\_CLKSOURCE\_\_: specifies the I2C4 clock source. This parameter can be one of the following values:
  - RCC\_I2C4CLKSOURCE\_PCLK1: PCLK1 selected as I2C4 clock
  - RCC\_I2C4CLKSOURCE\_HSI: HSI selected as I2C4 clock
  - RCC\_I2C4CLKSOURCE\_SYSCLK: System Clock selected as I2C4 clock

**\_\_HAL\_RCC\_GET\_I2C4\_SOURCE****Description:**

- macro to get the I2C4 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_I2C4CLKSOURCE\_PCLK1: PCLK1 selected as I2C4 clock
  - RCC\_I2C4CLKSOURCE\_HSI: HSI selected as I2C4 clock
  - RCC\_I2C4CLKSOURCE\_SYSCLK: System Clock selected as I2C4 clock

**\_\_HAL\_RCC\_USART1\_CONFIG****Description:**

- Macro to configure the USART1 clock (USART1CLK).

**Parameters:**

- \_\_USART1\_CLKSOURCE\_\_: specifies the USART1 clock source. This parameter can be one of the following

values:

- RCC\_USART1CLKSOURCE\_PCLK2: PCLK2 selected as USART1 clock
- RCC\_USART1CLKSOURCE\_HSI: HSI selected as USART1 clock
- RCC\_USART1CLKSOURCE\_SYSC LK: System Clock selected as USART1 clock
- RCC\_USART1CLKSOURCE\_LSE: LSE selected as USART1 clock

#### \_\_HAL\_RCC\_GET\_USART1\_SOURCE

**Description:**

- macro to get the USART1 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_USART1CLKSOURCE\_PCLK2: PCLK2 selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_HSI: HSI selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_SYSC LK: System Clock selected as USART1 clock
  - RCC\_USART1CLKSOURCE\_LSE: LSE selected as USART1 clock

#### \_\_HAL\_RCC\_USART2\_CONFIG

**Description:**

- Macro to configure the USART2 clock (USART2CLK).

**Parameters:**

- \_\_USART2\_CLKSOURCE\_\_: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC\_USART2CLKSOURCE\_PCLK1: PCLK1 selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_HSI: HSI selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_SYSC LK: System Clock selected as USART2 clock
  - RCC\_USART2CLKSOURCE\_LSE: LSE selected as USART2 clock

#### \_\_HAL\_RCC\_GET\_USART2\_SOURCE

**Description:**

- macro to get the USART2 clock source.

**Return value:**

- The: clock source can be one of the

following values:

- RCC\_USART2CLKSOURCE\_PCLK1: PCLK1 selected as USART2 clock
- RCC\_USART2CLKSOURCE\_HSI: HSI selected as USART2 clock
- RCC\_USART2CLKSOURCE\_SYSC LK: System Clock selected as USART2 clock
- RCC\_USART2CLKSOURCE\_LSE: LSE selected as USART2 clock

#### \_\_HAL\_RCC\_USART3\_CONFIG

##### Description:

- Macro to configure the USART3 clock (USART3CLK).

##### Parameters:

- \_\_USART3\_CLKSOURCE\_\_: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC\_USART3CLKSOURCE\_PCLK1: PCLK1 selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_HSI: HSI selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_SYSC LK: System Clock selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_LSE: LSE selected as USART3 clock

#### \_\_HAL\_RCC\_GET\_USART3\_SOURCE

##### Description:

- macro to get the USART3 clock source.

##### Return value:

- The: clock source can be one of the following values:
  - RCC\_USART3CLKSOURCE\_PCLK1: PCLK1 selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_HSI: HSI selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_SYSC LK: System Clock selected as USART3 clock
  - RCC\_USART3CLKSOURCE\_LSE: LSE selected as USART3 clock

#### \_\_HAL\_RCC\_UART4\_CONFIG

##### Description:

- Macro to configure the UART4 clock (UART4CLK).

##### Parameters:

### \_\_HAL\_RCC\_GET\_UART4\_SOURCE

- `__UART4_CLKSOURCE__`: specifies the UART4 clock source. This parameter can be one of the following values:
  - `RCC_UART4CLKSOURCE_PCLK1`: PCLK1 selected as UART4 clock
  - `RCC_UART4CLKSOURCE_HSI`: HSI selected as UART4 clock
  - `RCC_UART4CLKSOURCE_SYSCLK`: System Clock selected as UART4 clock
  - `RCC_UART4CLKSOURCE_LSE`: LSE selected as UART4 clock

#### Description:

- macro to get the UART4 clock source.

#### Return value:

- The: clock source can be one of the following values:
  - `RCC_UART4CLKSOURCE_PCLK1`: PCLK1 selected as UART4 clock
  - `RCC_UART4CLKSOURCE_HSI`: HSI selected as UART4 clock
  - `RCC_UART4CLKSOURCE_SYSCLK`: System Clock selected as UART4 clock
  - `RCC_UART4CLKSOURCE_LSE`: LSE selected as UART4 clock

### \_\_HAL\_RCC\_UART5\_CONFIG

#### Description:

- Macro to configure the UART5 clock (UART5CLK).

#### Parameters:

- `__UART5_CLKSOURCE__`: specifies the UART5 clock source. This parameter can be one of the following values:
  - `RCC_UART5CLKSOURCE_PCLK1`: PCLK1 selected as UART5 clock
  - `RCC_UART5CLKSOURCE_HSI`: HSI selected as UART5 clock
  - `RCC_UART5CLKSOURCE_SYSCLK`: System Clock selected as UART5 clock
  - `RCC_UART5CLKSOURCE_LSE`: LSE selected as UART5 clock

### \_\_HAL\_RCC\_GET\_UART5\_SOURCE

#### Description:

- macro to get the UART5 clock source.

#### Return value:

- The: clock source can be one of the following values:
  - `RCC_UART5CLKSOURCE_PCLK1`

`__HAL_RCC_USART6_CONFIG`

- : PCLK1 selected as UART5 clock
- `RCC_USART5CLKSOURCE_HSI`: HSI selected as UART5 clock
- `RCC_USART5CLKSOURCE_SYSCLK`: System Clock selected as UART5 clock
- `RCC_USART5CLKSOURCE_LSE`: LSE selected as UART5 clock

**Description:**

- Macro to configure the USART6 clock (USART6CLK).

**Parameters:**

- `__USART6_CLKSOURCE__`: specifies the USART6 clock source. This parameter can be one of the following values:
  - `RCC_USART6CLKSOURCE_PCLK1`: PCLK1 selected as USART6 clock
  - `RCC_USART6CLKSOURCE_HSI`: HSI selected as USART6 clock
  - `RCC_USART6CLKSOURCE_SYSCLOCK`: System Clock selected as USART6 clock
  - `RCC_USART6CLKSOURCE_LSE`: LSE selected as USART6 clock

`__HAL_RCC_GET_USART6_SOURCE`**Description:**

- macro to get the USART6 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - `RCC_USART6CLKSOURCE_PCLK1`: PCLK1 selected as USART6 clock
  - `RCC_USART6CLKSOURCE_HSI`: HSI selected as USART6 clock
  - `RCC_USART6CLKSOURCE_SYSCLOCK`: System Clock selected as USART6 clock
  - `RCC_USART6CLKSOURCE_LSE`: LSE selected as USART6 clock

`__HAL_RCC_UART7_CONFIG`**Description:**

- Macro to configure the UART7 clock (UART7CLK).

**Parameters:**

- `__UART7_CLKSOURCE__`: specifies the UART7 clock source. This parameter can be one of the following values:

- RCC\_UART7CLKSOURCE\_PCLK1 : PCLK1 selected as UART7 clock
- RCC\_UART7CLKSOURCE\_HSI: HSI selected as UART7 clock
- RCC\_UART7CLKSOURCE\_SYSCLK: System Clock selected as UART7 clock
- RCC\_UART7CLKSOURCE\_LSE: LSE selected as UART7 clock

#### \_\_HAL\_RCC\_GET\_UART7\_SOURCE

##### Description:

- macro to get the UART7 clock source.

##### Return value:

- The: clock source can be one of the following values:
  - RCC\_UART7CLKSOURCE\_PCLK1 : PCLK1 selected as UART7 clock
  - RCC\_UART7CLKSOURCE\_HSI: HSI selected as UART7 clock
  - RCC\_UART7CLKSOURCE\_SYSCLK: System Clock selected as UART7 clock
  - RCC\_UART7CLKSOURCE\_LSE: LSE selected as UART7 clock

#### \_\_HAL\_RCC\_UART8\_CONFIG

##### Description:

- Macro to configure the UART8 clock (UART8CLK).

##### Parameters:

- \_\_UART8\_CLKSOURCE\_\_: specifies the UART8 clock source. This parameter can be one of the following values:
  - RCC\_UART8CLKSOURCE\_PCLK1 : PCLK1 selected as UART8 clock
  - RCC\_UART8CLKSOURCE\_HSI: HSI selected as UART8 clock
  - RCC\_UART8CLKSOURCE\_SYSCLK: System Clock selected as UART8 clock
  - RCC\_UART8CLKSOURCE\_LSE: LSE selected as UART8 clock

#### \_\_HAL\_RCC\_GET\_UART8\_SOURCE

##### Description:

- macro to get the UART8 clock source.

##### Return value:

- The: clock source can be one of the following values:
  - RCC\_UART8CLKSOURCE\_PCLK1 : PCLK1 selected as UART8 clock
  - RCC\_UART8CLKSOURCE\_HSI: HSI selected as UART8 clock



- RCC\_UART8CLKSOURCE\_SYSCLK: System Clock selected as UART8 clock
- RCC\_UART8CLKSOURCE\_LSE: LSE selected as UART8 clock

**\_\_HAL\_RCC\_LPTIM1\_CONFIG****Description:**

- Macro to configure the LPTIM1 clock (LPTIM1CLK).

**Parameters:**

- **\_\_LPTIM1\_CLKSOURCE\_\_**: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC\_LPTIM1CLKSOURCE\_PCLK: PCLK selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_HSI: HSI selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_LSI: LSI selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_LSE: LSE selected as LPTIM1 clock

**\_\_HAL\_RCC\_GET\_LPTIM1\_SOURCE****Description:**

- macro to get the LPTIM1 clock source.

**Return value:**

- The: clock source can be one of the following values:
  - RCC\_LPTIM1CLKSOURCE\_PCLK: PCLK selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_HSI: HSI selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_LSI: LSI selected as LPTIM1 clock
  - RCC\_LPTIM1CLKSOURCE\_LSE: LSE selected as LPTIM1 clock

**\_\_HAL\_RCC\_CEC\_CONFIG****Description:**

- Macro to configure the CEC clock (CECCLK).

**Parameters:**

- **\_\_CEC\_CLKSOURCE\_\_**: specifies the CEC clock source. This parameter can be one of the following values:
  - RCC\_CECCLKSOURCE\_LSE: LSE selected as CEC clock
  - RCC\_CECCLKSOURCE\_HSI: HSI selected as CEC clock

**\_\_HAL\_RCC\_GET\_CEC\_SOURCE****Description:**

- macro to get the CEC clock source.

`__HAL_RCC_CLK48_CONFIG`**Return value:**

- The: clock source can be one of the following values:
  - `RCC_CECCLKSOURCE_LSE`: LSE selected as CEC clock
  - `RCC_CECCLKSOURCE_HSI`: HSI selected as CEC clock

**Description:**

- Macro to configure the CLK48 source (`CLK48CLK`).

**Parameters:**

- `__CLK48_SOURCE__`: specifies the CLK48 clock source. This parameter can be one of the following values:
  - `RCC_CLK48SOURCE_PLL`: PLL selected as CLK48 source
  - `RCC_CLK48SOURCE_PLSAI1`: PLLSAI1 selected as CLK48 source

`__HAL_RCC_GET_CLK48_SOURCE`**Description:**

- macro to get the CLK48 source.

**Return value:**

- The: clock source can be one of the following values:
  - `RCC_CLK48SOURCE_PLL`: PLL used as CLK48 source
  - `RCC_CLK48SOURCE_PLSAI1`: PLLSAI1 used as CLK48 source

`__HAL_RCC_SDMMC1_CONFIG`**Description:**

- Macro to configure the SDMMC1 clock (`SDMMC1CLK`).

**Parameters:**

- `__SDMMC1_CLKSOURCE__`: specifies the SDMMC1 clock source. This parameter can be one of the following values:
  - `RCC_SDMMC1CLKSOURCE_CLK48`: CLK48 selected as SDMMC clock
  - `RCC_SDMMC1CLKSOURCE_SYSCLK`: SYSCLK selected as SDMMC clock

`__HAL_RCC_GET_SDMMC1_SOURCE`**Description:**

- macro to get the SDMMC1 clock source.

**Return value:**

- The: clock source can be one of the following values:

- RCC\_SDMMC1CLKSOURCE\_CLK48: CLK48 selected as SDMMC1 clock
- RCC\_SDMMC1CLKSOURCE\_SYSCLK: SYSCLK selected as SDMMC1 clock

***RCCEX Force Release Peripheral Reset***

\_\_HAL\_RCC\_DMA2\_FORCE\_RESET  
\_\_HAL\_RCC\_DMA2D\_FORCE\_RESET  
\_\_HAL\_RCC\_ETHMAC\_FORCE\_RESET  
\_\_HAL\_RCC\_USB\_OTG\_HS\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOA\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOB\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOC\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOD\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOE\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOF\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOG\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOH\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOI\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOJ\_FORCE\_RESET  
\_\_HAL\_RCC\_GPIOK\_FORCE\_RESET  
\_\_HAL\_RCC\_DMA2\_RELEASE\_RESET  
\_\_HAL\_RCC\_DMA2D\_RELEASE\_RESET  
\_\_HAL\_RCC\_ETHMAC\_RELEASE\_RESET  
\_\_HAL\_RCC\_USB\_OTG\_HS\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOA\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOB\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOC\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOD\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOE\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOF\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOG\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOH\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOI\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOJ\_RELEASE\_RESET  
\_\_HAL\_RCC\_GPIOK\_RELEASE\_RESET  
\_\_HAL\_RCC\_AHB2\_FORCE\_RESET

\_\_HAL\_RCC\_DCMI\_FORCE\_RESET  
\_\_HAL\_RCC\_RNG\_FORCE\_RESET  
\_\_HAL\_RCC\_USB\_OTG\_FS\_FORCE\_RESET  
\_\_HAL\_RCC\_AHB2\_RELEASE\_RESET  
\_\_HAL\_RCC\_DCMI\_RELEASE\_RESET  
\_\_HAL\_RCC\_RNG\_RELEASE\_RESET  
\_\_HAL\_RCC\_USB\_OTG\_FS\_RELEASE\_RESET  
\_\_HAL\_RCC\_Cryp\_FORCE\_RESET  
\_\_HAL\_RCC\_HASH\_FORCE\_RESET  
\_\_HAL\_RCC\_Cryp\_RELEASE\_RESET  
\_\_HAL\_RCC\_HASH\_RELEASE\_RESET  
\_\_HAL\_RCC\_AHB3\_FORCE\_RESET  
\_\_HAL\_RCC\_FMC\_FORCE\_RESET  
\_\_HAL\_RCC\_QSPI\_FORCE\_RESET  
\_\_HAL\_RCC\_AHB3\_RELEASE\_RESET  
\_\_HAL\_RCC\_FMC\_RELEASE\_RESET  
\_\_HAL\_RCC\_QSPI\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM2\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM3\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM4\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM5\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM6\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM7\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM12\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM13\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM14\_FORCE\_RESET  
\_\_HAL\_RCC\_LPTIM1\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI2\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI3\_FORCE\_RESET  
\_\_HAL\_RCC\_SPDIFRX\_FORCE\_RESET  
\_\_HAL\_RCC\_USART2\_FORCE\_RESET  
\_\_HAL\_RCC\_USART3\_FORCE\_RESET  
\_\_HAL\_RCC\_UART4\_FORCE\_RESET  
\_\_HAL\_RCC\_UART5\_FORCE\_RESET  
\_\_HAL\_RCC\_I2C1\_FORCE\_RESET  
\_\_HAL\_RCC\_I2C2\_FORCE\_RESET

\_\_HAL\_RCC\_I2C3\_FORCE\_RESET  
\_\_HAL\_RCC\_I2C4\_FORCE\_RESET  
\_\_HAL\_RCC\_CAN1\_FORCE\_RESET  
\_\_HAL\_RCC\_CAN2\_FORCE\_RESET  
\_\_HAL\_RCC\_CEC\_FORCE\_RESET  
\_\_HAL\_RCC\_DAC\_FORCE\_RESET  
\_\_HAL\_RCC\_UART7\_FORCE\_RESET  
\_\_HAL\_RCC\_UART8\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM2\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM3\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM4\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM5\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM6\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM7\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM12\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM13\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM14\_RELEASE\_RESET  
\_\_HAL\_RCC\_LPTIM1\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI2\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI3\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPDIFRX\_RELEASE\_RESET  
\_\_HAL\_RCC\_USART2\_RELEASE\_RESET  
\_\_HAL\_RCC\_USART3\_RELEASE\_RESET  
\_\_HAL\_RCC\_UART4\_RELEASE\_RESET  
\_\_HAL\_RCC\_UART5\_RELEASE\_RESET  
\_\_HAL\_RCC\_I2C1\_RELEASE\_RESET  
\_\_HAL\_RCC\_I2C2\_RELEASE\_RESET  
\_\_HAL\_RCC\_I2C3\_RELEASE\_RESET  
\_\_HAL\_RCC\_I2C4\_RELEASE\_RESET  
\_\_HAL\_RCC\_CAN1\_RELEASE\_RESET  
\_\_HAL\_RCC\_CAN2\_RELEASE\_RESET  
\_\_HAL\_RCC\_CEC\_RELEASE\_RESET  
\_\_HAL\_RCC\_DAC\_RELEASE\_RESET  
\_\_HAL\_RCC\_UART7\_RELEASE\_RESET  
\_\_HAL\_RCC\_UART8\_RELEASE\_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\_SDMMC1\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI1\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI4\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM9\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM10\_FORCE\_RESET  
\_\_HAL\_RCC\_TIM11\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI5\_FORCE\_RESET  
\_\_HAL\_RCC\_SPI6\_FORCE\_RESET  
\_\_HAL\_RCC\_SAI1\_FORCE\_RESET  
\_\_HAL\_RCC\_SAI2\_FORCE\_RESET  
\_\_HAL\_RCC\_LTDC\_FORCE\_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\_SDMMC1\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI1\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI4\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM9\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM10\_RELEASE\_RESET  
\_\_HAL\_RCC\_TIM11\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI5\_RELEASE\_RESET  
\_\_HAL\_RCC\_SPI6\_RELEASE\_RESET  
\_\_HAL\_RCC\_SAI1\_RELEASE\_RESET  
\_\_HAL\_RCC\_SAI2\_RELEASE\_RESET  
\_\_HAL\_RCC\_LTDC\_RELEASE\_RESET

**RCCEX I2C1 Clock Source**

RCC\_I2C1CLKSOURCE\_PCLK1  
RCC\_I2C1CLKSOURCE\_SYSCLK  
RCC\_I2C1CLKSOURCE\_HSI

**RCCEX I2C2 Clock Source**

RCC\_I2C2CLKSOURCE\_PCLK1

RCC\_I2C2CLKSOURCE\_SYSCLK

RCC\_I2C2CLKSOURCE\_HSI

***RCCEx I2C3 Clock Source***

RCC\_I2C3CLKSOURCE\_PCLK1

RCC\_I2C3CLKSOURCE\_SYSCLK

RCC\_I2C3CLKSOURCE\_HSI

***RCCEx I2C4 Clock Source***

RCC\_I2C4CLKSOURCE\_PCLK1

RCC\_I2C4CLKSOURCE\_SYSCLK

RCC\_I2C4CLKSOURCE\_HSI

***RCCEx I2S Clock Source***

RCC\_I2SCLKSOURCE\_PLLI2S

RCC\_I2SCLKSOURCE\_EXT

***RCC Private macros to check input parameters***

IS\_RCC\_PERIPHCLK

IS\_RCC\_PLLI2SN\_VALUE

IS\_RCC\_PLLI2SP\_VALUE

IS\_RCC\_PLLI2SQ\_VALUE

IS\_RCC\_PLLI2SR\_VALUE

IS\_RCC\_PLLSAIN\_VALUE

IS\_RCC\_PLLSAIP\_VALUE

IS\_RCC\_PLLSAIQ\_VALUE

IS\_RCC\_PLLSAIR\_VALUE

IS\_RCC\_PLLSAI\_DIVQ\_VALUE

IS\_RCC\_PLLI2S\_DIVQ\_VALUE

IS\_RCC\_PLLSAI\_DIVR\_VALUE

IS\_RCC\_I2SCLKSOURCE

IS\_RCC\_SAI1CLKSOURCE

IS\_RCC\_SAI2CLKSOURCE

IS\_RCC\_SDMMC1CLKSOURCE

IS\_RCC\_CECCLKSOURCE

IS\_RCC\_USART1CLKSOURCE

IS\_RCC\_USART2CLKSOURCE

IS\_RCC\_USART3CLKSOURCE

IS\_RCC\_UART4CLKSOURCE

IS\_RCC\_UART5CLKSOURCE  
IS\_RCC\_USART6CLKSOURCE  
IS\_RCC\_UART7CLKSOURCE  
IS\_RCC\_UART8CLKSOURCE  
IS\_RCC\_I2C1CLKSOURCE  
IS\_RCC\_I2C2CLKSOURCE  
IS\_RCC\_I2C3CLKSOURCE  
IS\_RCC\_I2C4CLKSOURCE  
IS\_RCC\_LPTIM1CLK  
IS\_RCC\_CLK48SOURCE  
IS\_RCC\_TIMPRES

***RCCEX LPTIM1 Clock Source***

RCC\_LPTIM1CLKSOURCE\_PCLK  
RCC\_LPTIM1CLKSOURCE\_LSI  
RCC\_LPTIM1CLKSOURCE\_HSI  
RCC\_LPTIM1CLKSOURCE\_LSE

***RCCEX Peripheral Clock Enable/Disable***

\_\_HAL\_RCC\_BKPSRAM\_CLK\_ENABLE

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

\_\_HAL\_RCC\_DTCMRAMEN\_CLK\_ENABLE  
\_\_HAL\_RCC\_DMA2\_CLK\_ENABLE  
\_\_HAL\_RCC\_DMA2D\_CLK\_ENABLE  
\_\_HAL\_RCC\_USB\_OTG\_HS\_CLK\_ENABLE  
\_\_HAL\_RCC\_USB\_OTG\_HS\_ULPI\_CLK\_ENABLE  
\_\_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\_GPIOJ\_CLK\_ENABLE  
\_\_HAL\_RCC\_GPIOK\_CLK\_ENABLE  
\_\_HAL\_RCC\_BKPSRAM\_CLK\_DISABLE  
\_\_HAL\_RCC\_DTCMRAMEN\_CLK\_DISABLE  
\_\_HAL\_RCC\_DMA2\_CLK\_DISABLE  
\_\_HAL\_RCC\_DMA2D\_CLK\_DISABLE  
\_\_HAL\_RCC\_USB\_OTG\_HS\_CLK\_DISABLE  
\_\_HAL\_RCC\_USB\_OTG\_HS\_ULPI\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOA\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOB\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOC\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOD\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOE\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOF\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOG\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOH\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOI\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOJ\_CLK\_DISABLE  
\_\_HAL\_RCC\_GPIOK\_CLK\_DISABLE  
\_\_HAL\_RCC\_ETHMAC\_CLK\_ENABLE  
\_\_HAL\_RCC\_ETHMACTX\_CLK\_ENABLE  
\_\_HAL\_RCC\_ETHMACRX\_CLK\_ENABLE  
\_\_HAL\_RCC\_ETHMACPTP\_CLK\_ENABLE  
\_\_HAL\_RCC\_ETH\_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\_DISABLE  
\_\_HAL\_RCC\_DCMI\_CLK\_ENABLE  
  
\_\_HAL\_RCC\_RNG\_CLK\_ENABLE  
\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_ENABLE

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

\_\_HAL\_RCC\_DCMI\_CLK\_DISABLE  
\_\_HAL\_RCC\_RNG\_CLK\_DISABLE  
\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_DISABLE  
\_\_HAL\_RCC\_Cryp\_CLK\_ENABLE  
\_\_HAL\_RCC\_HASH\_CLK\_ENABLE  
\_\_HAL\_RCC\_Cryp\_CLK\_DISABLE  
\_\_HAL\_RCC\_HASH\_CLK\_DISABLE  
\_\_HAL\_RCC\_FMC\_CLK\_ENABLE

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

\_\_HAL\_RCC\_QSPI\_CLK\_ENABLE  
\_\_HAL\_RCC\_FMC\_CLK\_DISABLE  
\_\_HAL\_RCC\_QSPI\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM2\_CLK\_ENABLE

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

\_\_HAL\_RCC\_TIM3\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM4\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM5\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM6\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM7\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM12\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM13\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM14\_CLK\_ENABLE  
\_\_HAL\_RCC\_LPTIM1\_CLK\_ENABLE  
\_\_HAL\_RCC\_SPI2\_CLK\_ENABLE  
\_\_HAL\_RCC\_SPI3\_CLK\_ENABLE  
\_\_HAL\_RCC\_SPDIFRX\_CLK\_ENABLE  
\_\_HAL\_RCC\_USART2\_CLK\_ENABLE  
\_\_HAL\_RCC\_USART3\_CLK\_ENABLE  
\_\_HAL\_RCC\_UART4\_CLK\_ENABLE  
\_\_HAL\_RCC\_UART5\_CLK\_ENABLE

\_\_HAL\_RCC\_I2C1\_CLK\_ENABLE  
\_\_HAL\_RCC\_I2C2\_CLK\_ENABLE  
\_\_HAL\_RCC\_I2C3\_CLK\_ENABLE  
\_\_HAL\_RCC\_I2C4\_CLK\_ENABLE  
\_\_HAL\_RCC\_CAN1\_CLK\_ENABLE  
\_\_HAL\_RCC\_CAN2\_CLK\_ENABLE  
\_\_HAL\_RCC\_CEC\_CLK\_ENABLE  
\_\_HAL\_RCC\_DAC\_CLK\_ENABLE  
\_\_HAL\_RCC\_UART7\_CLK\_ENABLE  
\_\_HAL\_RCC\_UART8\_CLK\_ENABLE  
\_\_HAL\_RCC\_TIM2\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM3\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM4\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM5\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM6\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM7\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM12\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM13\_CLK\_DISABLE  
\_\_HAL\_RCC\_TIM14\_CLK\_DISABLE  
\_\_HAL\_RCC\_LPTIM1\_CLK\_DISABLE  
\_\_HAL\_RCC\_SPI2\_CLK\_DISABLE  
\_\_HAL\_RCC\_SPI3\_CLK\_DISABLE  
\_\_HAL\_RCC\_SPDIFRX\_CLK\_DISABLE  
\_\_HAL\_RCC\_USART2\_CLK\_DISABLE  
\_\_HAL\_RCC\_USART3\_CLK\_DISABLE  
\_\_HAL\_RCC\_UART4\_CLK\_DISABLE  
\_\_HAL\_RCC\_UART5\_CLK\_DISABLE  
\_\_HAL\_RCC\_I2C1\_CLK\_DISABLE  
\_\_HAL\_RCC\_I2C2\_CLK\_DISABLE  
\_\_HAL\_RCC\_I2C3\_CLK\_DISABLE  
\_\_HAL\_RCC\_I2C4\_CLK\_DISABLE  
\_\_HAL\_RCC\_CAN1\_CLK\_DISABLE  
\_\_HAL\_RCC\_CAN2\_CLK\_DISABLE  
\_\_HAL\_RCC\_CEC\_CLK\_DISABLE  
\_\_HAL\_RCC\_DAC\_CLK\_DISABLE  
\_\_HAL\_RCC\_UART7\_CLK\_DISABLE

\_\_HAL\_RCC\_UART8\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM1\_CLK\_ENABLE

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

\_\_HAL\_RCC\_TIM8\_CLK\_ENABLE

\_\_HAL\_RCC\_USART1\_CLK\_ENABLE

\_\_HAL\_RCC\_USART6\_CLK\_ENABLE

\_\_HAL\_RCC\_ADC1\_CLK\_ENABLE

\_\_HAL\_RCC\_ADC2\_CLK\_ENABLE

\_\_HAL\_RCC\_ADC3\_CLK\_ENABLE

\_\_HAL\_RCC\_SDMMC1\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI1\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI4\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM9\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM10\_CLK\_ENABLE

\_\_HAL\_RCC\_TIM11\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI5\_CLK\_ENABLE

\_\_HAL\_RCC\_SPI6\_CLK\_ENABLE

\_\_HAL\_RCC\_SAI1\_CLK\_ENABLE

\_\_HAL\_RCC\_SAI2\_CLK\_ENABLE

\_\_HAL\_RCC\_LTDC\_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\_SDMMC1\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI1\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI4\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM9\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM10\_CLK\_DISABLE

\_\_HAL\_RCC\_TIM11\_CLK\_DISABLE

\_\_HAL\_RCC\_SPI5\_CLK\_DISABLE  
\_\_HAL\_RCC\_SPI6\_CLK\_DISABLE  
\_\_HAL\_RCC\_SAI1\_CLK\_DISABLE  
\_\_HAL\_RCC\_SAI2\_CLK\_DISABLE  
\_\_HAL\_RCC\_LTDC\_CLK\_DISABLE

**Peripheral Clock Enable Disable Status**

\_\_HAL\_RCC\_BKPSRAM\_IS\_CLK\_ENABLED

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

\_\_HAL\_RCC\_DTCMRAMEN\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_DMA2\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_DMA2D\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_USB\_OTG\_HS\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_USB\_OTG\_HS\_ULPI\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOA\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOB\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOC\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOD\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOE\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOF\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOG\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOH\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOI\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOJ\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_GPIOK\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_BKPSRAM\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_DTCMRAMEN\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_DMA2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_DMA2D\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USB\_OTG\_HS\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USB\_OTG\_HS\_ULPI\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOA\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOB\_IS\_CLK\_DISABLED

\_\_HAL\_RCC\_GPIOC\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOD\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOE\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOF\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOG\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOH\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOI\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOJ\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_GPIOK\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ETHMAC\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ETHMACTX\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ETHMACRX\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ETHMACPTP\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ETH\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ETHMAC\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ETHMACTX\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ETHMACRX\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ETHMACPTP\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ETH\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_DCMI\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_RNG\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_USB\_OTG\_FS\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_DCMI\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_RNG\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USB\_IS\_OTG\_FS\_CLK\_DISABLED  
\_\_HAL\_RCC\_Cryp\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_HASH\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_Cryp\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_HASH\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_FMC\_IS\_CLK\_ENABLED

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

**Notes:**

- After reset, the peripheral clock (used for registers

read/write access) is disabled and the application software has to enable this clock before using it.

\_\_HAL\_RCC\_QSPI\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_FMC\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_QSPI\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM2\_IS\_CLK\_ENABLED

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

\_\_HAL\_RCC\_TIM3\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM4\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM5\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM6\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM7\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM12\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM13\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM14\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_LPTIM1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SPI2\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SPI3\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SPDIFRX\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_USART2\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_USART3\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_UART4\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_UART5\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_I2C1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_I2C2\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_I2C3\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_I2C4\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_CAN1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_CAN2\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_CEC\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_DAC\_IS\_CLK\_ENABLED

\_\_HAL\_RCC\_UART7\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_UART8\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM3\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM4\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM5\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM6\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM7\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM12\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM13\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM14\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_LPTIM1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SPI2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SPI3\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SPDIFRX\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USART2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USART3\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_UART4\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_UART5\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_I2C1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_I2C2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_I2C3\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_I2C4\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_CAN1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_CAN2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_CEC\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_DAC\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_UART7\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_UART8\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM1\_IS\_CLK\_ENABLED  
  
\_\_HAL\_RCC\_TIM8\_IS\_CLK\_ENABLED

**Notes:**

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.



\_\_HAL\_RCC\_USART1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_USART6\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ADC1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ADC2\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_ADC3\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SDMMC1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SPI1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SPI4\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM9\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM10\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM11\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SPI5\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SPI6\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SAI1\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_SAI2\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_LTDC\_IS\_CLK\_ENABLED  
\_\_HAL\_RCC\_TIM1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM8\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USART1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_USART6\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ADC1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ADC2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_ADC3\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SDMMC1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SPI1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SPI4\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM9\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM10\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_TIM11\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SPI5\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SPI6\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SAI1\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_SAI2\_IS\_CLK\_DISABLED  
\_\_HAL\_RCC\_LTDC\_IS\_CLK\_DISABLED  
***RCCEX Peripheral Clock Sleep Enable Disable***  
\_\_HAL\_RCC\_FLITF\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_AXI\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_SRAM1\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_SRAM2\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_BKPSRAM\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_DTCM\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_DMA2\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_DMA2D\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_ETHMAC\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_ETHMACTX\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_ETHMACRX\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_ETHMACPTP\_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\_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\_GPIOJ\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_GPIOK\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_FLITF\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_AXI\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_SRAM1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_SRAM2\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_BKPSRAM\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_DTCM\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_DMA2\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_DMA2D\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_ETHMAC\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_ETHMACTX\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_ETHMACRX\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_ETHMACPTP\_CLK\_SLEEP\_DISABLE

\_\_HAL\_RCC\_USB\_OTG\_HS\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_USB\_OTG\_HS\_ULPI\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOA\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOB\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOC\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOD\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOE\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOF\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOG\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOH\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOI\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOJ\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_GPIOK\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_DCM1\_CLK\_SLEEP\_ENABLE

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

\_\_HAL\_RCC\_DCM1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_RNG\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_RNG\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_USB\_OTG\_FS\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_Cryp\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_HASH\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_Cryp\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_HASH\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_FMC\_CLK\_SLEEP\_ENABLE

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP

mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

\_\_HAL\_RCC\_FMC\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_QSPI\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_QSPI\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM2\_CLK\_SLEEP\_ENABLE

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

\_\_HAL\_RCC\_TIM3\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM4\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM5\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM6\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM7\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM12\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM13\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM14\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_LPTIM1\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_SPI2\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_SPI3\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_SPDIFRX\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_USART2\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_USART3\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_UART4\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_UART5\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_I2C1\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_I2C2\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_I2C3\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_I2C4\_CLK\_SLEEP\_ENABLE

\_\_HAL\_RCC\_CAN1\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_CAN2\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_CEC\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_DAC\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_UART7\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_UART8\_CLK\_SLEEP\_ENABLE  
\_\_HAL\_RCC\_TIM2\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM3\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM4\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM5\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM6\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM7\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM12\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM13\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM14\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_LPTIM1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_SPI2\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_SPI3\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_SPDIFRX\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_USART2\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_USART3\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_UART4\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_UART5\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_I2C1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_I2C2\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_I2C3\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_I2C4\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_CAN1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_CAN2\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_CEC\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_DAC\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_UART7\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_UART8\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_TIM1\_CLK\_SLEEP\_ENABLE

**Notes:**

- Peripheral clock gating in SLEEP mode can be used to further reduce power

consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

```
__HAL_RCC_TIM8_CLK_SLEEP_ENABLE
__HAL_RCC_USART1_CLK_SLEEP_ENABLE
__HAL_RCC_USART6_CLK_SLEEP_ENABLE
__HAL_RCC_ADC1_CLK_SLEEP_ENABLE
__HAL_RCC_ADC2_CLK_SLEEP_ENABLE
__HAL_RCC_ADC3_CLK_SLEEP_ENABLE
__HAL_RCC_SDMMC1_CLK_SLEEP_ENABLE
__HAL_RCC_SPI1_CLK_SLEEP_ENABLE
__HAL_RCC_SPI4_CLK_SLEEP_ENABLE
__HAL_RCC_TIM9_CLK_SLEEP_ENABLE
__HAL_RCC_TIM10_CLK_SLEEP_ENABLE
__HAL_RCC_TIM11_CLK_SLEEP_ENABLE
__HAL_RCC_SPI5_CLK_SLEEP_ENABLE
__HAL_RCC_SPI6_CLK_SLEEP_ENABLE
__HAL_RCC_SAI1_CLK_SLEEP_ENABLE
__HAL_RCC_SAI2_CLK_SLEEP_ENABLE
__HAL_RCC_LTDC_CLK_SLEEP_ENABLE
__HAL_RCC_TIM1_CLK_SLEEP_DISABLE
__HAL_RCC_TIM8_CLK_SLEEP_DISABLE
__HAL_RCC_USART1_CLK_SLEEP_DISABLE
__HAL_RCC_USART6_CLK_SLEEP_DISABLE
__HAL_RCC_ADC1_CLK_SLEEP_DISABLE
__HAL_RCC_ADC2_CLK_SLEEP_DISABLE
__HAL_RCC_ADC3_CLK_SLEEP_DISABLE
__HAL_RCC_SDMMC1_CLK_SLEEP_DISABLE
__HAL_RCC_SPI1_CLK_SLEEP_DISABLE
__HAL_RCC_SPI4_CLK_SLEEP_DISABLE
__HAL_RCC_TIM9_CLK_SLEEP_DISABLE
__HAL_RCC_TIM10_CLK_SLEEP_DISABLE
__HAL_RCC_TIM11_CLK_SLEEP_DISABLE
__HAL_RCC_SPI5_CLK_SLEEP_DISABLE
```

\_\_HAL\_RCC\_SPI6\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_SAI1\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_SAI2\_CLK\_SLEEP\_DISABLE  
\_\_HAL\_RCC\_LTDC\_CLK\_SLEEP\_DISABLE

***RCC Periph Clock Selection***

RCC\_PERIPHCLK\_I2S  
RCC\_PERIPHCLK\_LTDC  
RCC\_PERIPHCLK\_TIM  
RCC\_PERIPHCLK\_RTC  
RCC\_PERIPHCLK\_USART1  
RCC\_PERIPHCLK\_USART2  
RCC\_PERIPHCLK\_USART3  
RCC\_PERIPHCLK\_UART4  
RCC\_PERIPHCLK\_UART5  
RCC\_PERIPHCLK\_USART6  
RCC\_PERIPHCLK\_UART7  
RCC\_PERIPHCLK\_UART8  
RCC\_PERIPHCLK\_I2C1  
RCC\_PERIPHCLK\_I2C2  
RCC\_PERIPHCLK\_I2C3  
RCC\_PERIPHCLK\_I2C4  
RCC\_PERIPHCLK\_LPTIM1  
RCC\_PERIPHCLK\_SAI1  
RCC\_PERIPHCLK\_SAI2  
RCC\_PERIPHCLK\_CLK48  
RCC\_PERIPHCLK\_CEC  
RCC\_PERIPHCLK\_SDMMC1  
RCC\_PERIPHCLK\_SPDIFRX  
RCC\_PERIPHCLK\_PLLI2S

***RCCEx PLLSAIP Clock Divider***

RCC\_PLLSAIP\_DIV2  
RCC\_PLLSAIP\_DIV4  
RCC\_PLLSAIP\_DIV6  
RCC\_PLLSAIP\_DIV8

***RCCEx PLLSAI DIVR***

RCC\_PLLSAIDIVR\_2

RCC\_PLLSAIDIVR\_4

RCC\_PLLSAIDIVR\_8

RCC\_PLLSAIDIVR\_16

***RCCEX Private Defines***

PLLI2S\_TIMEOUT\_VALUE

PLLSAI\_TIMEOUT\_VALUE

***RCCEX SAI1 Clock Source***

RCC\_SAI1CLKSOURCE\_PLLSAI

RCC\_SAI1CLKSOURCE\_PLLI2S

RCC\_SAI1CLKSOURCE\_PIN

***RCCEX SAI2 Clock Source***

RCC\_SAI2CLKSOURCE\_PLLSAI

RCC\_SAI2CLKSOURCE\_PLLI2S

RCC\_SAI2CLKSOURCE\_PIN

***RCCEX SDMMC1 Clock Source***

RCC\_SDMMC1CLKSOURCE\_CLK48

RCC\_SDMMC1CLKSOURCE\_SYSCLK

***RCCEX TIM Prescaler Selection***

RCC\_TIMPRES\_DESACTIVATED

RCC\_TIMPRES\_ACTIVATED

***RCCEX UART4 Clock Source***

RCC\_UART4CLKSOURCE\_PCLK1

RCC\_UART4CLKSOURCE\_SYSCLK

RCC\_UART4CLKSOURCE\_HSI

RCC\_UART4CLKSOURCE\_LSE

***RCCEX UART5 Clock Source***

RCC\_UART5CLKSOURCE\_PCLK1

RCC\_UART5CLKSOURCE\_SYSCLK

RCC\_UART5CLKSOURCE\_HSI

RCC\_UART5CLKSOURCE\_LSE

***RCCEX UART7 Clock Source***

RCC\_UART7CLKSOURCE\_PCLK1

RCC\_UART7CLKSOURCE\_SYSCLK

RCC\_UART7CLKSOURCE\_HSI

RCC\_UART7CLKSOURCE\_LSE

***RCCEX UART8 Clock Source***



RCC\_UART8CLKSOURCE\_PCLK1

RCC\_UART8CLKSOURCE\_SYSCLK

RCC\_UART8CLKSOURCE\_HSI

RCC\_UART8CLKSOURCE\_LSE

***RCCEX USART1 Clock Source***

RCC\_USART1CLKSOURCE\_PCLK2

RCC\_USART1CLKSOURCE\_SYSCLK

RCC\_USART1CLKSOURCE\_HSI

RCC\_USART1CLKSOURCE\_LSE

***RCCEX USART2 Clock Source***

RCC\_USART2CLKSOURCE\_PCLK1

RCC\_USART2CLKSOURCE\_SYSCLK

RCC\_USART2CLKSOURCE\_HSI

RCC\_USART2CLKSOURCE\_LSE

***RCCEX USART3 Clock Source***

RCC\_USART3CLKSOURCE\_PCLK1

RCC\_USART3CLKSOURCE\_SYSCLK

RCC\_USART3CLKSOURCE\_HSI

RCC\_USART3CLKSOURCE\_LSE

***RCCEX USART6 Clock Source***

RCC\_USART6CLKSOURCE\_PCLK2

RCC\_USART6CLKSOURCE\_SYSCLK

RCC\_USART6CLKSOURCE\_HSI

RCC\_USART6CLKSOURCE\_LSE

## 45 HAL RNG Generic Driver

### 45.1 RNG Firmware driver registers structures

#### 45.1.1 RNG\_HandleTypeDef

##### Data Fields

- *RNG\_TypeDef \* Instance*
- *uint32\_t RandomNumber*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_RNG\_StateTypeDef State*

##### Field Documentation

- *RNG\_TypeDef\* RNG\_HandleTypeDef::Instance*  
Register base address
- *uint32\_t RNG\_HandleTypeDef::RandomNumber*  
Last Generated random number
- *HAL\_LockTypeDef RNG\_HandleTypeDef::Lock*  
RNG locking object
- *\_\_IO HAL\_RNG\_StateTypeDef RNG\_HandleTypeDef::State*  
RNG communication state

### 45.2 RNG Firmware driver API description

#### 45.2.1 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.

#### 45.2.2 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\\_Msplnit\(\)](#)
- [HAL\\_RNG\\_MspDelnit\(\)](#)

### 45.2.3 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\(\)](#)

### 45.2.4 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\(\)](#)

### 45.2.5 HAL\_RNG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)</b>
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>

### 45.2.6 HAL\_RNG\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)</b>
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>

### 45.2.7 HAL\_RNG\_Msplnit

Function Name	<b>void HAL_RNG_Msplnit (RNG_HandleTypeDef * hrng)</b>
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

- None

### 45.2.8 HAL\_RNG\_MspDeInit

Function Name **void HAL\_RNG\_MspDeInit (RNG\_HandleTypeDef \* hrng)**

Function Description DeInitializes the RNG MSP.

Parameters

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

Return values

- None

### 45.2.9 HAL\_RNG\_GenerateRandomNumber

Function Name **HAL\_StatusTypeDef HAL\_RNG\_GenerateRandomNumber (RNG\_HandleTypeDef \* hrng, uint32\_t \* random32bit)**

Function Description Generates a 32-bit random number.

Parameters

- **hrng**: pointer to a RNG\_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit**: pointer to generated random number variable if successful.

Return values

- HAL status

Notes

- Each time the random number data is read the RNG\_FLAG\_DRDY flag is automatically cleared.

### 45.2.10 HAL\_RNG\_GenerateRandomNumber\_IT

Function Name **HAL\_StatusTypeDef HAL\_RNG\_GenerateRandomNumber\_IT (RNG\_HandleTypeDef \* hrng)**

Function Description Generates a 32-bit random number in interrupt mode.

Parameters

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

Return values

- HAL status

### 45.2.11 HAL\_RNG\_IRQHandler

Function Name **void HAL\_RNG\_IRQHandler (RNG\_HandleTypeDef \* hrng)**

Function Description Handles RNG interrupt request.

Parameters

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

Return values

- None

Notes

- In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using `__HAL_RNG_CLEAR_IT()`. The clock error has no impact on the previously generated random numbers, and the RNG\_DR register contents can be used.

- In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG\_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using `__HAL_RNG_CLEAR_IT()`, then disable and enable the RNG peripheral to reinitialize and restart the RNG.
- User-written `HAL_RNG_ErrorCallback()` API is called once whether SEIS or CEIS are set.

### 45.2.12 HAL\_RNG\_GetRandomNumber

Function Name	<b>uint32_t HAL_RNG_GetRandomNumber (RNG_HandleTypeDef * hrng)</b>
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 <code>RNG_HandleTypeDef</code> structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Random value</li> </ul>

### 45.2.13 HAL\_RNG\_GetRandomNumber\_IT

Function Name	<b>uint32_t HAL_RNG_GetRandomNumber_IT (RNG_HandleTypeDef * hrng)</b>
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 <code>RNG_HandleTypeDef</code> 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>

### 45.2.14 HAL\_RNG\_ReadLastRandomNumber

Function Name	<b>uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)</b>
Function Description	Read latest generated random number.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b>: pointer to a <code>RNG_HandleTypeDef</code> structure that contains the configuration information for RNG.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• random value</li> </ul>

### 45.2.15 HAL\_RNG\_ReadyDataCallback

Function Name	<b>void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)</b>
Function Description	Data Ready callback in non-blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrng</b>: pointer to a <code>RNG_HandleTypeDef</code> 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>

**45.2.16 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>

**45.2.17 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>

**45.3 RNG Firmware driver defines****45.3.1 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***

<b>__HAL_RNG_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset RNG handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__:</b> RNG Handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
<b>__HAL_RNG_ENABLE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Enables the RNG peripheral.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__:</b> RNG Handle</li> </ul> <b>Return value:</b>

**\_\_HAL\_RNG\_DISABLE**

- None

**Description:**

- Disables the RNG peripheral.

**Parameters:**

- **\_\_HANDLE\_\_**: RNG Handle

**Return value:**

- None

**\_\_HAL\_RNG\_GET\_FLAG****Description:**

- Check the selected RNG flag status.

**Parameters:**

- **\_\_HANDLE\_\_**: RNG Handle
- **\_\_FLAG\_\_**: RNG flag This parameter can be one of the following values:
  - **RNG\_FLAG\_DRDY**: Data ready
  - **RNG\_FLAG\_CECS**: Clock error current status
  - **RNG\_FLAG\_SECS**: Seed error current status

**Return value:**

- The: new state of **\_\_FLAG\_\_** (SET or RESET).

**\_\_HAL\_RNG\_CLEAR\_FLAG****Description:**

- Clears the selected RNG flag status.

**Parameters:**

- **\_\_HANDLE\_\_**: RNG handle
- **\_\_FLAG\_\_**: RNG flag to clear

**Return value:**

- None

**Notes:**

- **WARNING:** This is a dummy macro for HAL code alignment, flags **RNG\_FLAG\_DRDY**, **RNG\_FLAG\_CECS** and **RNG\_FLAG\_SECS** are read-only.

**\_\_HAL\_RNG\_ENABLE\_IT****Description:**

- Enables the RNG interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: RNG Handle

**Return value:**

- None

**\_\_HAL\_RNG\_DISABLE\_IT****Description:**

**\_\_HAL\_RNG\_GET\_IT**

- Disables the RNG interrupts.

**Parameters:**

- **\_\_HANDLE\_\_**: RNG Handle

**Return value:**

- None

**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).

**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**



## 46 HAL RTC Generic Driver

### 46.1 RTC Firmware driver registers structures

#### 46.1.1 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 = 0x7FFF
- *uint32\_t RTC\_InitTypeDef::OutPut*  
Specifies which signal will be routed to the RTC output. This parameter can be a value of [RTCEx\\_Output\\_selection\\_Definitions](#)
- *uint32\_t RTC\_InitTypeDef::OutPutPolarity*  
Specifies the polarity of the output signal. This parameter can be a value of [RTC\\_Output\\_Polarity\\_Definitions](#)
- *uint32\_t RTC\_InitTypeDef::OutPutType*  
Specifies the RTC Output Pin mode. This parameter can be a value of [RTC\\_Output\\_Type\\_ALARM\\_OUT](#)

#### 46.1.2 RTC\_TimeTypeDef

##### Data Fields

- *uint8\_t Hours*
- *uint8\_t Minutes*
- *uint8\_t Seconds*
- *uint32\_t SubSeconds*
- *uint8\_t TimeFormat*
- *uint32\_t DayLightSaving*
- *uint32\_t StoreOperation*

### Field Documentation

- ***uint8\_t RTC\_TimeTypeDef::Hours***  
Specifies the RTC Time Hour. This parameter must be a number between Min\_Data = 0 and Max\_Data = 12 if the RTC\_HourFormat\_12 is selected. This parameter must be a number between Min\_Data = 0 and Max\_Data = 23 if the RTC\_HourFormat\_24 is selected
- ***uint8\_t RTC\_TimeTypeDef::Minutes***  
Specifies the RTC Time Minutes. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59
- ***uint8\_t RTC\_TimeTypeDef::Seconds***  
Specifies the RTC Time Seconds. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59
- ***uint32\_t RTC\_TimeTypeDef::SubSeconds***  
Specifies the RTC Time SubSeconds. This parameter must be a number between Min\_Data = 0 and Max\_Data = 59
- ***uint8\_t RTC\_TimeTypeDef::TimeFormat***  
Specifies the RTC AM/PM Time. This parameter can be a value of [RTC\\_AM\\_PM\\_Definitions](#)
- ***uint32\_t RTC\_TimeTypeDef::DayLightSaving***  
Specifies RTC\_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of [RTC\\_DayLightSaving\\_Definitions](#)
- ***uint32\_t RTC\_TimeTypeDef::StoreOperation***  
Specifies RTC\_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of [RTC\\_StoreOperation\\_Definitions](#)

## 46.1.3 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

#### 46.1.4 RTC\_AlarmTypeDef

##### Data Fields

- *RTC\_TimeTypeDef AlarmTime*
- *uint32\_t AlarmMask*
- *uint32\_t AlarmSubSecondMask*
- *uint32\_t AlarmDateWeekDaySel*
- *uint8\_t AlarmDateWeekDay*
- *uint32\_t Alarm*

##### Field Documentation

- *RTC\_TimeTypeDef RTC\_AlarmTypeDef::AlarmTime*  
Specifies the RTC Alarm Time members
- *uint32\_t RTC\_AlarmTypeDef::AlarmMask*  
Specifies the RTC Alarm Masks. This parameter can be a value of [RTC\\_AlarmMask\\_Definitions](#)
- *uint32\_t RTC\_AlarmTypeDef::AlarmSubSecondMask*  
Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of [RTC\\_Alarm\\_Sub\\_Seconds\\_Masks\\_Definitions](#)
- *uint32\_t RTC\_AlarmTypeDef::AlarmDateWeekDaySel*  
Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of [RTC\\_AlarmDateWeekDay\\_Definitions](#)
- *uint8\_t RTC\_AlarmTypeDef::AlarmDateWeekDay*  
Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of [RTC\\_WeekDay\\_Definitions](#)
- *uint32\_t RTC\_AlarmTypeDef::Alarm*  
Specifies the alarm . This parameter can be a value of [RTC\\_Alarms\\_Definitions](#)

#### 46.1.5 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

## 46.2 RTC Firmware driver API description

### 46.2.1 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
4. PC1 can be used as the RTC\_AF3 pin

### 46.2.2 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.

### 46.2.3 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__PWR_CLK_ENABLE()` function.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.
- Select the RTC clock source using the `__HAL_RCC_RTC_CONFIG()` function.

- Enable RTC Clock using the `__HAL_RCC_RTC_ENABLE()` function.

#### 46.2.4 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.

#### 46.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

#### 46.2.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
  - A 7-bit asynchronous prescaler and a 13-bit synchronous prescaler.
  - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, `RTC_WPR`.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the

initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.

4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC\_TR and RTC\_DR shadow registers. The HAL\_RTC\_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- [HAL\\_RTC\\_Init\(\)](#)
- [HAL\\_RTC\\_DeInit\(\)](#)
- [HAL\\_RTC\\_MspltInit\(\)](#)
- [HAL\\_RTC\\_MspDeInit\(\)](#)

#### 46.2.7 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\(\)](#)

#### 46.2.8 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\(\)](#)

#### 46.2.9 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\(\)](#)

#### 46.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- [HAL\\_RTC\\_GetState\(\)](#)

#### 46.2.11 HAL\_RTC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)</b>
Function Description	Initializes 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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
<b>46.2.12 HAL_RTC_DeInit</b>	
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	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>This function doesn't reset the RTC Backup Data registers.</li> </ul>
<b>46.2.13 HAL_RTC_MspltInit</b>	
Function Name	<b>void HAL_RTC_MspltInit (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	<ul style="list-style-type: none"> <li>None</li> </ul>
<b>46.2.14 HAL_RTC_MspDeInit</b>	
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	<ul style="list-style-type: none"> <li>None</li> </ul>
<b>46.2.15 HAL_RTC_SetTime</b>	
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: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**46.2.16 HAL\_RTC\_GetTime**

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)</b>
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: FORMAT_BIN: Binary data formatFORMAT_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>

**46.2.17 HAL\_RTC\_SetDate**

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * 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: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**46.2.18 HAL\_RTC\_GetDate**

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * 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: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**46.2.19 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: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 46.2.20 HAL\_RTC\_SetAlarm\_IT

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetAlarm_IT</b> (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
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: FORMAT_BIN: Binary data format 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>• The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).</li> <li>• The HAL_RTC_SetTime() must be called before enabling the Alarm feature.</li> </ul>

#### 46.2.21 HAL\_RTC\_DeactivateAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_DeactivateAlarm</b> (RTC_HandleTypeDef * hrtc, uint32_t Alarm)
Function Description	Deactive 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: AlarmA RTC_ALARM_B: AlarmB</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 46.2.22 HAL\_RTC\_GetAlarm

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetAlarm</b> (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)
Function Description	Gets the RTC Alarm value and masks.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sAlarm</b>: Pointer to Date structure</li> </ul>

- **Alarm:** Specifies the Alarm. This parameter can be one of the following values: RTC\_ALARM\_A: AlarmA, RTC\_ALARM\_B: AlarmB
  - **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT\_BIN: Binary data format, FORMAT\_BCD: BCD data format
- Return values
- HAL status

#### 46.2.23 HAL\_RTC\_AlarmIRQHandler

- Function Name** void HAL\_RTC\_AlarmIRQHandler (RTC\_HandleTypeDef \* hrtc)
- Function Description** This function handles Alarm interrupt request.
- Parameters**
- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- Return values**
- None

#### 46.2.24 HAL\_RTC\_AlarmAEventCallback

- Function Name** void HAL\_RTC\_AlarmAEventCallback (RTC\_HandleTypeDef \* hrtc)
- Function Description** Alarm A callback.
- Parameters**
- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- Return values**
- None

#### 46.2.25 HAL\_RTC\_PollForAlarmAEvent

- Function Name** HAL\_StatusTypeDef HAL\_RTC\_PollForAlarmAEvent (RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)
- Function Description** This function handles AlarmA Polling request.
- Parameters**
- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
  - **Timeout:** Timeout duration
- Return values**
- HAL status

#### 46.2.26 HAL\_RTC\_WaitForSynchro

- Function Name** HAL\_StatusTypeDef HAL\_RTC\_WaitForSynchro (RTC\_HandleTypeDef \* hrtc)
- Function Description** Waits until the RTC Time and Date registers (RTC\_TR and RTC\_DR) are synchronized with RTC APB clock.
- Parameters**
- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- Return values**
- HAL status
- Notes**
- The RTC Resynchronization mode is write protected, use the \_\_HAL\_RTC\_WRITEPROTECTION\_DISABLE() before calling this function.

- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC\_TR and RTC\_DR shadow registers.

#### 46.2.27 HAL\_RTC\_GetState

Function Name	<b>HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)</b>
Function Description	Returns the RTC state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

#### 46.2.28 HAL\_RTC\_Init

Function Name	<b>HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)</b>
Function Description	Initializes 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 46.2.29 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	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• This function doesn't reset the RTC Backup Data registers.</li> </ul>

#### 46.2.30 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	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 46.2.31 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	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 46.2.32 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: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 46.2.33 HAL\_RTC\_GetTime

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)</b>
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: FORMAT_BIN: Binary data formatFORMAT_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>

#### 46.2.34 HAL\_RTC\_SetDate

Function Name	<b>HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * 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: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**46.2.35 HAL\_RTC\_GetDate**

Function Name	<b>HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * 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: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**46.2.36 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: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**46.2.37 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: FORMAT_BIN: Binary data format 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>• The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).</li> <li>• The HAL_RTC_SetTime() must be called before enabling the Alarm feature.</li> </ul>

**46.2.38 HAL\_RTC\_DeactivateAlarm**

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

---

**(RTC\_HandleTypeDef \* hrtc, uint32\_t Alarm)**

Function Description	Deactive 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: AlarmARTC_ALARM_B: AlarmB</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 46.2.39 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 contains the configuration information for RTC.</li> <li>• <b>sAlarm</b>: Pointer to Date structure</li> <li>• <b>Alarm</b>: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmARTC_ALARM_B: AlarmB</li> <li>• <b>Format</b>: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 46.2.40 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>

#### 46.2.41 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>

#### 46.2.42 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>

### 46.2.43 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 wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.</li> </ul>

### 46.2.44 HAL\_RTC\_GetState

Function Name	<b>HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)</b>
Function Description	Returns the RTC state.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

## 46.3 RTC Firmware driver defines

### 46.3.1 RTC

#### *RTC Alarm Date WeekDay Definitions*

RTC\_ALARMDATEWEEKDAYSEL\_DATE

RTC\_ALARMDATEWEEKDAYSEL\_WEEKDAY

#### *RTC Alarm Mask Definitions*

RTC\_ALARMMAK\_NONE

RTC\_ALARMMAK\_DATEWEEKDAY

RTC\_ALARMMAK\_HOURS

RTC\_ALARM\_MASK\_MINUTES

RTC\_ALARM\_MASK\_SECONDS

RTC\_ALARM\_MASK\_ALL

### **RTC Alarms Definitions**

RTC\_ALARM\_A

RTC\_ALARM\_B

### **RTC Alarm Sub Seconds Masks Definitions**

RTC\_ALARMSSUBSECONDMASK\_ALL All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

RTC\_ALARMSSUBSECONDMASK\_SS14\_1 SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.

RTC\_ALARMSSUBSECONDMASK\_SS14\_2 SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_3 SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_4 SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_5 SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_6 SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_7 SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_8 SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_9 SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_10 SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_11 SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_12 SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14\_13 SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared

RTC\_ALARMSSUBSECONDMASK\_SS14 SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared

RTC\_ALARMSSUBSECONDMASK\_NONE SS[14:0] are compared and must match to activate alarm.

### **RTC AM PM Definitions**

RTC\_HOURFORMAT12\_AM

RTC\_HOURFORMAT12\_PM



**RTC DayLight Saving Definitions**

RTC\_DAYLIGHTSAVING\_SUB1H

RTC\_DAYLIGHTSAVING\_ADD1H

RTC\_DAYLIGHTSAVING\_NONE

**RTC Exported Macros**

\_\_HAL\_RTC\_RESET\_HANDLE\_STATE

**Description:**

- Reset RTC handle state.

**Parameters:**

- \_\_HANDLE\_\_: 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.

**Return value:**

- None

**Description:**

- Enable the RTC ALARMB peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Disable the RTC ALARMB peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**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

**Description:**

`__HAL_RTC_ALARMB_ENABLE`

`__HAL_RTC_ALARMB_DISABLE`

`__HAL_RTC_ALARM_ENABLE_IT`

`__HAL_RTC_ALARM_DISABLE_IT`

- 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

**Description:**

- 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

**Description:**

- 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`

`__HAL_RTC_ALARM_GET_IT`

`__HAL_RTC_ALARM_GET_FLAG`

- RTC\_FLAG\_ALRAWF
- RTC\_FLAG\_ALRBWF

**Return value:**

- None

**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

`__HAL_RTC_ALARM_CLEAR_FLAG`

`__HAL_RTC_ALARM_GET_IT_SOURCE`

`__HAL_RTC_ALARM_EXTI_ENABLE_IT`

__HAL_RTC_ALARM_EXTI_DISABLE_IT	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>• Disable interrupt on the RTC Alarm associated Exti line.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>• None</li></ul>
__HAL_RTC_ALARM_EXTI_ENABLE_EVENT	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>• Enable event on the RTC Alarm associated Exti line.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>• None.</li></ul>
__HAL_RTC_ALARM_EXTI_DISABLE_EVENT	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>• Disable event on the RTC Alarm associated Exti line.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>• None.</li></ul>
__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>• Enable falling edge trigger on the RTC Alarm associated Exti line.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>• None.</li></ul>
__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>• Disable falling edge trigger on the RTC Alarm associated Exti line.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>• None.</li></ul>
__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>• Enable rising edge trigger on the RTC Alarm associated Exti line.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>• None.</li></ul>
__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE	<p><b>Description:</b></p> <ul style="list-style-type: none"><li>• Disable rising edge trigger on the RTC Alarm associated Exti line.</li></ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"><li>• None.</li></ul>

\_\_HAL\_RTC\_ALARM\_EXTI\_ENABLE\_RISING\_FALLING\_EDGE

**Description:**

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

\_\_HAL\_RTC\_ALARM\_EXTI\_DISABLE\_RISING\_FALLING\_EDGE

**Description:**

- Disable rising & falling edge trigger on the RTC Alarm associated Exti line.

**Return value:**

- None.

\_\_HAL\_RTC\_ALARM\_EXTI\_GET\_FLAG

**Description:**

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

\_\_HAL\_RTC\_ALARM\_EXTI\_CLEAR\_FLAG

**Description:**

- Clear the RTC Alarm associated Exti line flag.

**Return value:**

- None.

\_\_HAL\_RTC\_ALARM\_EXTI\_GENERATE\_SWIT

**Description:**

- Generate a Software interrupt on RTC Alarm associated Exti line.

**Return value:**

- None.

**RTC Flags Definitions**

RTC\_FLAG\_RECALPF

RTC\_FLAG\_TAMP3F

RTC\_FLAG\_TAMP2F

RTC\_FLAG\_TAMP1F

RTC\_FLAG\_TSOVF

RTC\_FLAG\_TSF

RTC\_FLAG\_ITSF

RTC\_FLAG\_WUTF

RTC\_FLAG\_ALRBF

RTC\_FLAG\_ALRAF  
RTC\_FLAG\_INITF  
RTC\_FLAG\_RSF  
RTC\_FLAG\_INITS  
RTC\_FLAG\_SHPF  
RTC\_FLAG\_WUTWF  
RTC\_FLAG\_ALRBWF  
RTC\_FLAG\_ALRAWF

***RTC Hour Formats***

RTC\_HOURFORMAT\_24  
RTC\_HOURFORMAT\_12

***RTC Input Parameter Format Definitions***

RTC\_FORMAT\_BIN  
RTC\_FORMAT\_BCD

***RTC Interrupts Definitions***

RTC\_IT\_TS  
RTC\_IT\_WUT  
RTC\_IT\_ALRA  
RTC\_IT\_ALRB  
RTC\_IT\_TAMP  
RTC\_IT\_TAMP1  
RTC\_IT\_TAMP2  
RTC\_IT\_TAMP3

***RTC Private macros to check input parameters***

IS\_RTC\_HOUR\_FORMAT  
IS\_RTC\_OUTPUT\_POL  
IS\_RTC\_OUTPUT\_TYPE  
IS\_RTC\_ASYNCH\_PREDIV  
IS\_RTC\_SYNCH\_PREDIV  
IS\_RTC\_HOUR12  
IS\_RTC\_HOUR24  
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  
IS\_RTC\_ALARM\_SUB\_SECOND\_VALUE  
IS\_RTC\_ALARM\_SUB\_SECOND\_MASK

***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 Type ALARM OUT***

RTC\_OUTPUT\_TYPE\_OPENDRAIN  
RTC\_OUTPUT\_TYPE\_PUSHPULL

***RTC Private Constants***

RTC\_TR\_RESERVED\_MASK  
RTC\_DR\_RESERVED\_MASK  
RTC\_INIT\_MASK  
RTC\_RSF\_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

## 47 HAL RTC Extension Driver

### 47.1 RTCEX Firmware driver registers structures

#### 47.1.1 RTC\_TamperTypeDef

##### Data Fields

- *uint32\_t Tamper*
- *uint32\_t Interrupt*
- *uint32\_t Trigger*
- *uint32\_t NoErase*
- *uint32\_t MaskFlag*
- *uint32\_t Filter*
- *uint32\_t SamplingFrequency*
- *uint32\_t PrechargeDuration*
- *uint32\_t TamperPullUp*
- *uint32\_t TimeStampOnTamperDetection*

##### Field Documentation

- *uint32\_t RTC\_TamperTypeDef::Tamper*  
Specifies the Tamper Pin. This parameter can be a value of [RTCEX\\_Tamper\\_Pins\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::Interrupt*  
Specifies the Tamper Interrupt. This parameter can be a value of [RTCEX\\_Tamper\\_Interrupt\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::Trigger*  
Specifies the Tamper Trigger. This parameter can be a value of [RTCEX\\_Tamper\\_Trigger\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::NoErase*  
Specifies the Tamper no erase mode. This parameter can be a value of [RTCEX\\_Tamper\\_EraseBackUp\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::MaskFlag*  
Specifies the Tamper Flag masking. This parameter can be a value of [RTCEX\\_Tamper\\_MaskFlag\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::Filter*  
Specifies the RTC Filter Tamper. This parameter can be a value of [RTCEX\\_Tamper\\_Filter\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::SamplingFrequency*  
Specifies the sampling frequency. This parameter can be a value of [RTCEX\\_Tamper\\_Sampling\\_Frequencies\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::PrechargeDuration*  
Specifies the Precharge Duration . This parameter can be a value of [RTCEX\\_Tamper\\_Pin\\_Precharge\\_Duration\\_Definitions](#)
- *uint32\_t RTC\_TamperTypeDef::TamperPullUp*  
Specifies the Tamper PullUp . This parameter can be a value of [RTCEX\\_Tamper\\_Pull\\_UP\\_Definitions](#)

- ***uint32\_t RTC\_TamperTypeDef::TimeStampOnTamperDetection***  
Specifies the TimeStampOnTamperDetection. This parameter can be a value of [\*RTCEx\\_Tamper\\_TimeStampOnTamperDetection\\_Definitions\*](#)

## 47.2 RTCEx Firmware driver API description

### 47.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL\_RTC\_Init() function.

#### RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL\_RTC\_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer in interrupt mode using the HAL\_RTC\_SetWakeUpTimer\_IT() function.
- To read the RTC WakeUp Counter register, use the HAL\_RTC\_GetWakeUpTimer() function.

#### TimeStamp configuration

- Enables the RTC TimeStamp using the HAL\_RTC\_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL\_RTC\_SetTimeStamp\_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL\_RTC\_GetTimeStamp() function.

#### Internal TimeStamp configuration

- Enables the RTC internal TimeStamp using the HAL\_RTC\_SetInternalTimeStamp() function.
- To read the RTC TimeStamp Time and Date register, use the HAL\_RTC\_GetTimeStamp() function.

#### Tamper configuration

- Enable the RTC Tamper and Configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL\_RTC\_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL\_RTC\_SetTamper\_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the RTC\_TAMPCR register.

## Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL\_RTC\_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL\_RTC\_BKUPRead() function.

### 47.2.2 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\\_SetInternalTimeStamp\(\)](#)
- [HAL\\_RTCEX\\_DeactivateInternalTimeStamp\(\)](#)
- [HAL\\_RTCEX\\_GetTimeStamp\(\)](#)
- [HAL\\_RTCEX\\_SetTamper\(\)](#)
- [HAL\\_RTCEX\\_SetTamper\\_IT\(\)](#)
- [HAL\\_RTCEX\\_DeactivateTamper\(\)](#)
- [HAL\\_RTCEX\\_TamperTimeStampIRQHandler\(\)](#)
- [HAL\\_RTCEX\\_TimeStampEventCallback\(\)](#)
- [HAL\\_RTCEX\\_Tamper1EventCallback\(\)](#)
- [HAL\\_RTCEX\\_Tamper2EventCallback\(\)](#)
- [HAL\\_RTCEX\\_Tamper3EventCallback\(\)](#)
- [HAL\\_RTCEX\\_PollForTimeStampEvent\(\)](#)
- [HAL\\_RTCEX\\_PollForTamper1Event\(\)](#)
- [HAL\\_RTCEX\\_PollForTamper2Event\(\)](#)
- [HAL\\_RTCEX\\_PollForTamper3Event\(\)](#)

### 47.2.3 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\\_RTCEX\\_PollForWakeUpTimerEvent\(\)](#)

### 47.2.4 Extension Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC\_CALIB) Selection (1Hz or 512Hz).

- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:

- [HAL\\_RTCEX\\_BKUPWrite\(\)](#)
- [HAL\\_RTCEX\\_BKUPRead\(\)](#)
- [HAL\\_RTCEX\\_SetSmoothCalib\(\)](#)
- [HAL\\_RTCEX\\_SetSynchroShift\(\)](#)
- [HAL\\_RTCEX\\_SetCalibrationOutPut\(\)](#)
- [HAL\\_RTCEX\\_DeactivateCalibrationOutPut\(\)](#)
- [HAL\\_RTCEX\\_SetRefClock\(\)](#)
- [HAL\\_RTCEX\\_DeactivateRefClock\(\)](#)
- [HAL\\_RTCEX\\_EnableBypassShadow\(\)](#)
- [HAL\\_RTCEX\\_DisableBypassShadow\(\)](#)

### 47.2.5 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- [HAL\\_RTCEX\\_AlarmBEventCallback\(\)](#)
- [HAL\\_RTCEX\\_PollForAlarmBEvent\(\)](#)

### 47.2.6 HAL\_RTCEX\_SetTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_SetTimeStamp</b> (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)
Function Description	Sets TimeStamp.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>TimeStampEdge</b>: Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: 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_PC13: PC13 is selected as RTC TimeStamp Pin. RTC_TIMESTAMPPIN_PI8: PI8 is selected as RTC TimeStamp Pin. RTC_TIMESTAMPPIN_PC1: PC1 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>

### 47.2.7 HAL\_RTCEX\_SetTimeStamp\_IT



Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT</b> (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)
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_PC13: PC13 is selected as RTC TimeStamp Pin. RTC_TIMESTAMPPIN_PI8: PI8 is selected as RTC TimeStamp Pin. RTC_TIMESTAMPPIN_PC1: PC1 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>

#### 47.2.8 HAL\_RTCEx\_DeactivateTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp</b> (RTC_HandleTypeDef * hrtc)
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>

#### 47.2.9 HAL\_RTCEx\_SetInternalTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp</b> (RTC_HandleTypeDef * hrtc)
Function Description	Sets Internal 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>
Notes	<ul style="list-style-type: none"> <li>• This API must be called before enabling the internal TimeStamp feature.</li> </ul>

#### 47.2.10 HAL\_RTCEx\_DeactivateInternalTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp</b> (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates internal 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>   |

### 47.2.11 HAL\_RTCEX\_GetTimeStamp

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTCEX_GetTimeStamp</b><br>(RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)   |
| Function Description | Gets the RTC TimeStamp value.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>sTimeStamp:</b> Pointer to Time structure</li> <li>• <b>sTimeStampDate:</b> Pointer to Date structure</li> <li>• <b>Format:</b> specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |

### 47.2.12 HAL\_RTCEX\_SetTamper

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTCEX_SetTamper</b><br>(RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)  |
| 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>   |

### 47.2.13 HAL\_RTCEX\_SetTamper\_IT

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTCEX_SetTamper_IT</b><br>(RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)   |
| 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>   |

### 47.2.14 HAL\_RTCEX\_DeactivateTamper

- |               |   |
|---------------|---|
| Function Name | <b>HAL_StatusTypeDef HAL_RTCEX_DeactivateTamper</b> |
|---------------|---|

---

**(RTC\_HandleTypeDef \* hrtc, uint32\_t Tamper)**

Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>Tamper:</b> Selected tamper pin. This parameter can be RTC_Tamper_1 and/or RTC_TAMPER_2.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 47.2.15 HAL\_RTCEx\_TamperTimeStampIRQHandler

Function Name	<b>void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)</b>
Function Description	This function handles TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 47.2.16 HAL\_RTCEx\_TimeStampEventCallback

Function Name	<b>void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 47.2.17 HAL\_RTCEx\_Tamper1EventCallback

Function Name	<b>void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 47.2.18 HAL\_RTCEx\_Tamper2EventCallback

Function Name	<b>void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

#### 47.2.19 HAL\_RTCEx\_Tamper3EventCallback



Function Name	<b>void HAL_RTCEX_Tamper3EventCallback</b> (RTC_HandleTypeDef * hrtc)
---------------	--

Function Description	Tamper 3 callback.
----------------------	--------------------

Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> </ul>
------------	---

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

#### 47.2.20 HAL\_RTCEX\_PollForTimeStampEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_PollForTimeStampEvent</b> (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
---------------	--

Function Description	This function handles TimeStamp polling request.
----------------------	--

Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
------------	--

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

#### 47.2.21 HAL\_RTCEX\_PollForTamper1Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_PollForTamper1Event</b> (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
---------------	--

Function Description	This function handles Tamper1 Polling.
----------------------	--

Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
------------	--

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

#### 47.2.22 HAL\_RTCEX\_PollForTamper2Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_PollForTamper2Event</b> (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
---------------	--

Function Description	This function handles Tamper2 Polling.
----------------------	--

Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
------------	--

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

#### 47.2.23 HAL\_RTCEX\_PollForTamper3Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_PollForTamper3Event</b> (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
---------------	--

Function Description	This function handles Tamper3 Polling.
----------------------	--

Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc:</b> RTC handle</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
------------	---

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

#### 47.2.24 HAL\_RTCEX\_SetWakeUpTimer

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer</b> (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Sets wake up timer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>WakeUpCounter</b>: Wake up counter</li> <li>• <b>WakeUpClock</b>: Wake up clock</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>47.2.25 HAL_RTCEx_SetWakeUpTimer_IT</b>	
Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT</b> (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Sets wake up timer with interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>WakeUpCounter</b>: Wake up counter</li> <li>• <b>WakeUpClock</b>: Wake up clock</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>47.2.26 HAL_RTCEx_DeactivateWakeUpTimer</b>	
Function Name	<b>uint32_t HAL_RTCEx_DeactivateWakeUpTimer</b> (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates wake up timer counter.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
<b>47.2.27 HAL_RTCEx_GetWakeUpTimer</b>	
Function Name	<b>uint32_t HAL_RTCEx_GetWakeUpTimer</b> (RTC_HandleTypeDef * hrtc)
Function Description	Gets wake up timer counter.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• Counter value</li> </ul>
<b>47.2.28 HAL_RTCEx_WakeUpTimerIRQHandler</b>	
Function Name	<b>void HAL_RTCEx_WakeUpTimerIRQHandler</b> (RTC_HandleTypeDef * hrtc)
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

#### 47.2.29 HAL\_RTCEx\_WakeUpTimerEventCallback

Function Name **void HAL\_RTCEx\_WakeUpTimerEventCallback (RTC\_HandleTypeDef \* hrtc)**

Function Description Wake Up Timer callback.

Parameters

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

Return values

- None

#### 47.2.30 HAL\_RTCEx\_PollForWakeUpTimerEvent

Function Name **HAL\_StatusTypeDef HAL\_RTCEx\_PollForWakeUpTimerEvent (RTC\_HandleTypeDef \* hrtc, uint32\_t Timeout)**

Function Description This function handles Wake Up Timer Polling.

Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout:** Timeout duration

Return values

- HAL status

#### 47.2.31 HAL\_RTCEx\_BKUPWrite

Function Name **void HAL\_RTCEx\_BKUPWrite (RTC\_HandleTypeDef \* hrtc, uint32\_t BackupRegister, uint32\_t Data)**

Function Description Writes a data in a specified RTC Backup data register.

Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC\_BKP\_DRx where x can be from 0 to 19 to specify the register.
- **Data:** Data to be written in the specified RTC Backup data register.

Return values

- None

#### 47.2.32 HAL\_RTCEx\_BKUPRead

Function Name **uint32\_t HAL\_RTCEx\_BKUPRead (RTC\_HandleTypeDef \* hrtc, uint32\_t BackupRegister)**

Function Description Reads data from the specified RTC Backup data Register.

Parameters

- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC\_BKP\_DRx where x can be from 0 to 19 to specify the register.

Return values

- Read value

**47.2.33 HAL\_RTCEx\_SetSmoothCalib**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib</b> (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)
Function Description	Sets the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>SmoothCalibPeriod</b>: Select the Smooth Calibration Period. This parameter can be one of the following values : RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s. RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s. RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s.</li> <li>• <b>SmoothCalibPlusPulses</b>: Select to Set or reset the CALP bit. This parameter can be one of the following values: RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulses every 2<sup>11</sup> pulses. RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.</li> <li>• <b>SmoothCalibMinusPulsesValue</b>: Select the value of CALM[8:0] bits. This parameter can be any value from 0 to 0x000001FF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.</li> </ul>

**47.2.34 HAL\_RTCEx\_SetSynchroShift**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift</b> (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>ShiftAdd1S</b>: Select to add or not 1 second to the time calendar. This parameter can be one of the following values : RTC_SHIFTADD1S_SET: Add one second to the clock calendar. RTC_SHIFTADD1S_RESET: No effect.</li> <li>• <b>ShiftSubFS</b>: Select the number of Second Fractions to substitute. This parameter can be any value from 0 to 0x7FFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When REFCKON is set, firmware must not write to Shift control register.</li> </ul>

**47.2.35 HAL\_RTCEX\_SetCalibrationOutPut**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)</b>
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>CalibOutput:</b> Select the Calibration output Selection . This parameter can be one of the following values: RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**47.2.36 HAL\_RTCEX\_DeactivateCalibrationOutPut**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)</b>
Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**47.2.37 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>

**47.2.38 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>

**47.2.39 HAL\_RTCEX\_EnableBypassShadow**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_EnableBypassShadow (RTC_HandleTypeDef * hrtc)</b>
---------------	--

Function Description	Enables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.</li> </ul>

#### 47.2.40 HAL\_RTCEX\_DisableBypassShadow

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_DisableBypassShadow (RTC_HandleTypeDef * hrtc)</b>
Function Description	Disables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.</li> </ul>

#### 47.2.41 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>

#### 47.2.42 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>

#### 47.2.43 HAL\_RTCEX\_SetTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)</b>
Function Description	Sets TimeStamp.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>TimeStampEdge:</b> Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the</li> </ul>

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_TIMESTAMPPPIN_PC13`: PC13 is selected as RTC TimeStamp Pin. `RTC_TIMESTAMPPPIN_PI8`: PI8 is selected as RTC TimeStamp Pin. `RTC_TIMESTAMPPPIN_PC1`: PC1 is selected as RTC TimeStamp Pin.

Return values

- HAL status

Notes

- This API must be called before enabling the TimeStamp feature.

#### 47.2.44 HAL\_RTCEX\_SetTimeStamp\_IT

Function Name

**HAL\_StatusTypeDef HAL\_RTCEX\_SetTimeStamp\_IT**  
(**RTC\_HandleTypeDef \* hrtc, uint32\_t TimeStampEdge, uint32\_t RTC\_TimeStampPin**)

Function Description

Sets TimeStamp with Interrupt.

Parameters

- **hrtc:** 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_TIMESTAMPPPIN_PC13`: PC13 is selected as RTC TimeStamp Pin. `RTC_TIMESTAMPPPIN_PI8`: PI8 is selected as RTC TimeStamp Pin. `RTC_TIMESTAMPPPIN_PC1`: PC1 is selected as RTC TimeStamp Pin.

Return values

- HAL status

Notes

- This API must be called before enabling the TimeStamp feature.

#### 47.2.45 HAL\_RTCEX\_DeactivateTimeStamp

Function Name

**HAL\_StatusTypeDef HAL\_RTCEX\_DeactivateTimeStamp**  
(**RTC\_HandleTypeDef \* hrtc**)

Function Description

Deactivates TimeStamp.

Parameters

- **hrtc:** pointer to a `RTC_HandleTypeDef` structure that contains the configuration information for RTC.

Return values

- HAL status

#### 47.2.46 HAL\_RTCEX\_SetInternalTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp</b> (RTC_HandleTypeDef * hrtc)
Function Description	Sets Internal 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>
Notes	<ul style="list-style-type: none"> <li>This API must be called before enabling the internal TimeStamp feature.</li> </ul>

#### 47.2.47 HAL\_RTCEx\_DeactivateInternalTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp</b> (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates internal 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>

#### 47.2.48 HAL\_RTCEx\_GetTimeStamp

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp</b> (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)
Function Description	Gets the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>sTimeStamp:</b> Pointer to Time structure</li> <li><b>sTimeStampDate:</b> Pointer to Date structure</li> <li><b>Format:</b> specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

#### 47.2.49 HAL\_RTCEx\_SetTamper

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetTamper</b> (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
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 tamperers.</li> </ul>



**47.2.50 HAL\_RTCEX\_SetTamper\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_SetTamper_IT</b> (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
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>

**47.2.51 HAL\_RTCEX\_DeactivateTamper**

Function Name	<b>HAL_StatusTypeDef HAL_RTCEX_DeactivateTamper</b> (RTC_HandleTypeDef * hrtc, uint32_t Tamper)
Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li><b>Tamper</b>: Selected tamper pin. This parameter can be RTC_Tamper_1 and/or RTC_TAMPER_2.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**47.2.52 HAL\_RTCEX\_TamperTimeStampIRQHandler**

Function Name	<b>void HAL_RTCEX_TamperTimeStampIRQHandler</b> (RTC_HandleTypeDef * hrtc)
Function Description	This function handles TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**47.2.53 HAL\_RTCEX\_Tamper1EventCallback**

Function Name	<b>void HAL_RTCEX_Tamper1EventCallback</b> (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"> <li><b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**47.2.54 HAL\_RTCEX\_Tamper2EventCallback**

Function Name	<b>void HAL_RTCEX_Tamper2EventCallback</b> (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 2 callback.

Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 47.2.55 HAL\_RTCEx\_Tamper3EventCallback

Function Name	<b>void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	Tamper 3 callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: RTC handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 47.2.56 HAL\_RTCEx\_TimeStampEventCallback

Function Name	<b>void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)</b>
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 47.2.57 HAL\_RTCEx\_PollForTimeStampEvent

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles TimeStamp polling request.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 47.2.58 HAL\_RTCEx\_PollForTamper1Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles Tamper1 Polling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 47.2.59 HAL\_RTCEx\_PollForTamper2Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles Tamper2 Polling.

Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 47.2.60 HAL\_RTCEx\_PollForTamper3Event

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</b>
Function Description	This function handles Tamper3 Polling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: RTC handle</li> <li>• <b>Timeout</b>: Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 47.2.61 HAL\_RTCEx\_SetWakeUpTimer

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)</b>
Function Description	Sets wake up timer.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>WakeUpCounter</b>: Wake up counter</li> <li>• <b>WakeUpClock</b>: Wake up clock</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

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

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

**47.2.64 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	<ul style="list-style-type: none"> <li>Counter value</li> </ul>

**47.2.65 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	<ul style="list-style-type: none"> <li>None</li> </ul>

**47.2.66 HAL\_RTCEX\_WakeUpTimerEventCallback**

Function Name	<b>void HAL_RTCEX_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)</b>
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>

**47.2.67 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>

**47.2.68 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> </ul>

- **Data:** Data to be written in the specified RTC Backup data register.

Return values

- None

### 47.2.69 HAL\_RTCEX\_BKUPRead

**Function Name** `uint32_t HAL_RTCEX_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)`

**Function Description** Reads data from the specified RTC Backup data Register.

- Parameters**
- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
  - **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC\_BKP\_DRx where x can be from 0 to 19 to specify the register.

Return values

- Read value

### 47.2.70 HAL\_RTCEX\_SetSmoothCalib

**Function Name** `HAL_StatusTypeDef HAL_RTCEX_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)`

**Function Description** Sets the Smooth calibration parameters.

- Parameters**
- **hrtc:** pointer to a RTC\_HandleTypeDef structure that contains the configuration information for RTC.
  - **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be one of the following values :  
RTC\_SMOOTHCALIB\_PERIOD\_32SEC: The smooth calibration period is 32s.  
RTC\_SMOOTHCALIB\_PERIOD\_16SEC: The smooth calibration period is 16s.  
RTC\_SMOOTHCALIB\_PERIOD\_8SEC: The smooth calibration period is 8s.
  - **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values:  
RTC\_SMOOTHCALIB\_PLUSPULSES\_SET: Add one RTCCLK pulses every 2<sup>11</sup> pulses.  
RTC\_SMOOTHCALIB\_PLUSPULSES\_RESET: No RTCCLK pulses are added.
  - **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.

Return values

- HAL status

Notes

- To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB\_PLUSPULSES\_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

### 47.2.71 HAL\_RTCEX\_SetSynchroShift

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift</b> (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>ShiftAdd1S</b>: Select to add or not 1 second to the time calendar. This parameter can be one of the following values : RTC_SHIFTADD1S_SET: Add one second to the clock calendar.RTC_SHIFTADD1S_RESET: No effect.</li> <li>• <b>ShiftSubFS</b>: Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• When REFCKON is set, firmware must not write to Shift control register.</li> </ul>

#### 47.2.72 HAL\_RTCEx\_SetCalibrationOutPut

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut</b> (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> <li>• <b>CalibOutput</b>: Select the Calibration output Selection . This parameter can be one of the following values: RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 47.2.73 HAL\_RTCEx\_DeactivateCalibrationOutPut

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut</b> (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> <li>• <b>hrtc</b>: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 47.2.74 HAL\_RTCEx\_SetRefClock

Function Name	<b>HAL_StatusTypeDef HAL_RTCEx_SetRefClock</b> (RTC_HandleTypeDef * hrtc)
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>   |

#### 47.2.75 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>   |

#### 47.2.76 HAL\_RTCEX\_EnableBypassShadow

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTCEX_EnableBypassShadow (RTC_HandleTypeDef * hrtc)</b>   |
| Function Description | Enables the Bypass Shadow feature.   |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |
| Notes                | <ul style="list-style-type: none"> <li>• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.</li> </ul>         |

#### 47.2.77 HAL\_RTCEX\_DisableBypassShadow

- |                      |  |
|----------------------|--|
| Function Name        | <b>HAL_StatusTypeDef HAL_RTCEX_DisableBypassShadow (RTC_HandleTypeDef * hrtc)</b>  |
| Function Description | Disables the Bypass Shadow feature.  |
| Parameters           | <ul style="list-style-type: none"> <li>• <b>hrtc:</b> pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.</li> </ul> |
| Return values        | <ul style="list-style-type: none"> <li>• HAL status</li> </ul>   |
| Notes                | <ul style="list-style-type: none"> <li>• When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.</li> </ul>         |

#### 47.2.78 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>   |

#### 47.2.79 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>

## 47.3 RTCEx Firmware driver defines

### 47.3.1 RTCEx

#### *RTCEx Add 1 Second Parameter Definitions*

RTC\_SHIFTADD1S\_RESET

RTC\_SHIFTADD1S\_SET

#### *RTC Backup Registers Definitions*

RTC\_BKP\_DR0

RTC\_BKP\_DR1

RTC\_BKP\_DR2

RTC\_BKP\_DR3

RTC\_BKP\_DR4

RTC\_BKP\_DR5

RTC\_BKP\_DR6

RTC\_BKP\_DR7

RTC\_BKP\_DR8

RTC\_BKP\_DR9

RTC\_BKP\_DR10

RTC\_BKP\_DR11

RTC\_BKP\_DR12

RTC\_BKP\_DR13

RTC\_BKP\_DR14

RTC\_BKP\_DR15

RTC\_BKP\_DR16

RTC\_BKP\_DR17

RTC\_BKP\_DR18

RTC\_BKP\_DR19

RTC\_BKP\_DR20

RTC\_BKP\_DR21

RTC\_BKP\_DR22



RTC\_BKP\_DR23

RTC\_BKP\_DR24

RTC\_BKP\_DR25

RTC\_BKP\_DR26

RTC\_BKP\_DR27

RTC\_BKP\_DR28

RTC\_BKP\_DR29

RTC\_BKP\_DR30

RTC\_BKP\_DR31

***RTCEX Calib Output selection Definitions***

RTC\_CALIBOUTPUT\_512HZ

RTC\_CALIBOUTPUT\_1HZ

***RTCEX Exported Macros***

\_\_HAL\_RTC\_WAKEUPTIMER\_ENABLE

**Description:**

- Enable the RTC WakeUp Timer peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

\_\_HAL\_RTC\_WAKEUPTIMER\_DISABLE

**Description:**

- Disable the RTC WakeUp Timer peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

\_\_HAL\_RTC\_WAKEUPTIMER\_ENABLE\_IT

**Description:**

- Enable the RTC WakeUpTimer interrupt.

**Parameters:**

- `__HANDLE__`: specifies the RTC

### \_\_HAL\_RTC\_WAKEUPTIMER\_DISABLE\_IT

handle.

- **\_\_INTERRUPT\_\_**: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
  - **RTC\_IT\_WUT**: WakeUpTimer 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 interrupt

**Return value:**

- None

**Description:**

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the RTC handle.
- **\_\_INTERRUPT\_\_**: specifies the RTC WakeUpTimer interrupt sources to check. This parameter can be:

### \_\_HAL\_RTC\_WAKEUPTIMER\_GET\_IT

- RTC\_IT\_WUT: WakeUpTimer interrupt

**Return value:**

- None

**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

**Description:**

- Get the selected RTC WakeUpTimer's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
  - RTC\_FLAG\_WUTF
  - RTC\_FLAG\_WUTWF

**Return value:**

- None

**Description:**

- Clear the RTC Wake

`__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE`

`__HAL_RTC_WAKEUPTIMER_GET_FLAG`

`__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG`

Up timer's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
  - `RTC_FLAG_WUTF`

**Return value:**

- None

**Description:**

- Enable the RTC Tamper1 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Disable the RTC Tamper1 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Enable the RTC Tamper2 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

`__HAL_RTC_TAMPER1_ENABLE`

`__HAL_RTC_TAMPER1_DISABLE`

`__HAL_RTC_TAMPER2_ENABLE`

`__HAL_RTC_TAMPER2_DISABLE`

**Return value:**

- None

**Description:**

- Disable the RTC Tamper2 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Enable the RTC Tamper3 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Disable the RTC Tamper3 input detection.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Check whether the specified RTC Tamper interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC

`__HAL_RTC_TAMPER3_ENABLE`

`__HAL_RTC_TAMPER3_DISABLE`

`__HAL_RTC_TAMPER_GET_IT`

### \_\_HAL\_RTC\_TAMPER\_GET\_IT\_SOURCE

Tamper interrupt to check. This parameter can be:

- RTC\_IT\_TAMP: All tamper interrupts
- RTC\_IT\_TAMP1: Tamper1 interrupt
- RTC\_IT\_TAMP2: Tamper2 interrupt
- RTC\_IT\_TAMP3: Tamper3 interrupt

**Return value:**

- None

**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: All tamper interrupts
  - RTC\_IT\_TAMP1: Tamper1 interrupt
  - RTC\_IT\_TAMP2: Tamper2 interrupt
  - RTC\_IT\_TAMP3: Tamper3 interrupt

**Return value:**

- None

### \_\_HAL\_RTC\_TAMPER\_GET\_FLAG

**Description:**

- Get the selected RTC Tamper's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag is pending or not. This parameter can be:
  - `RTC_FLAG_TAM_P1F`: Tamper1 flag
  - `RTC_FLAG_TAM_P2F`: Tamper2 flag
  - `RTC_FLAG_TAM_P3F`: Tamper3 flag

**Return value:**

- None

**Description:**

- Clear the RTC Tamper's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag sources to clear. This parameter can be:
  - `RTC_FLAG_TAM_P1F`: Tamper1 flag
  - `RTC_FLAG_TAM_P2F`: Tamper2 flag
  - `RTC_FLAG_TAM_P3F`: Tamper3 flag

**Return value:**

- None

**Description:**

- Enable the RTC TimeStamp peripheral.

**Parameters:**

- `__HANDLE__`:

`__HAL_RTC_TAMPER_CLEAR_FLAG`

`__HAL_RTC_TIMESTAMP_ENABLE`

\_\_HAL\_RTC\_TIMESTAMP\_DISABLE

specifies the RTC handle.

**Return value:**

- None

**Description:**

- Disable the RTC TimeStamp peripheral.

**Parameters:**

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

**Return value:**

- None

**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

**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:

\_\_HAL\_RTC\_TIMESTAMP\_ENABLE\_IT

\_\_HAL\_RTC\_TIMESTAMP\_DISABLE\_IT



- RTC\_IT\_TS:  
TimeStamp  
interrupt

**Return value:**

- None

**Description:**

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt sources to check. This parameter can be:
  - RTC\_IT\_TS:  
TimeStamp  
interrupt

**Return value:**

- None

**Description:**

- 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

**Description:**

- Get the selected RTC TimeStamp's flag

`__HAL_RTC_TIMESTAMP_GET_IT`

`__HAL_RTC_TIMESTAMP_GET_IT_SOURCE`

`__HAL_RTC_TIMESTAMP_GET_FLAG`

status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
  - `RTC_FLAG_TSF`
  - `RTC_FLAG_TSOVF`

**Return value:**

- None

**Description:**

- Clear the RTC Time Stamp's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - `RTC_FLAG_TSF`
  - `RTC_FLAG_TSOVF`

**Return value:**

- None

**Description:**

- Enable the RTC internal TimeStamp peripheral.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Disable the RTC internal TimeStamp peripheral.

`__HAL_RTC_TIMESTAMP_CLEAR_FLAG`

`__HAL_RTC_INTERNAL_TIMESTAMP_ENABLE`

`__HAL_RTC_INTERNAL_TIMESTAMP_DISABLE`

\_\_HAL\_RTC\_INTERNAL\_TIMESTAMP\_GET\_FLAG

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.

**Return value:**

- None

**Description:**

- Get the selected RTC Internal Time Stamp's flag status.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Internal Time Stamp Flag is pending or not. This parameter can be:
  - `RTC_FLAG_ITSF`

**Return value:**

- None

\_\_HAL\_RTC\_INTERNAL\_TIMESTAMP\_CLEAR\_FLAG

**Description:**

- Clear the RTC Internal Time Stamp's pending flags.

**Parameters:**

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Internal Time Stamp Flag source to clear. This parameter can be:
  - `RTC_FLAG_ITSF`

**Return value:**

- None

\_\_HAL\_RTC\_CALIBRATION\_OUTPUT\_ENABLE

**Description:**

- Enable the RTC calibration output.

**Parameters:**

- `__HANDLE__`:

\_\_HAL\_RTC\_CALIBRATION\_OUTPUT\_DISABLE

specifies the RTC handle.

**Return value:**

- None

**Description:**

- Disable the calibration output.

**Parameters:**

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

**Return value:**

- None

**Description:**

- Enable the clock reference detection.

**Parameters:**

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

**Return value:**

- None

**Description:**

- Disable the clock reference detection.

**Parameters:**

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

**Return value:**

- None

**Description:**

- Get the selected RTC shift operation's flag status.

**Parameters:**

- \_\_HANDLE\_\_: specifies the RTC handle.
- \_\_FLAG\_\_: specifies the RTC shift operation Flag is

\_\_HAL\_RTC\_SHIFT\_GET\_FLAG

pending or not. This parameter can be:

- RTC\_FLAG\_SHP  
F

**Return value:**

- None

**Description:**

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None

**Description:**

- Disable interrupt on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None

**Description:**

- Enable event on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

**Description:**

- Disable event on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

**Description:**

- Enable falling edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

**Description:**

- Disable falling edge trigger on the RTC

\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_IT

\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_IT

\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_EVENT

\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_EVENT

\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_ENABLE\_FALLING\_EDGE

\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_DISABLE\_FALLING\_EDGE

WakeUp Timer  
associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE`

**Description:**

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE`

**Description:**

- Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE`

**Description:**

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE`

**Description:**

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_GET_FLAG`

**Description:**

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

`__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG`

**Description:**

- Clear the RTC

WakeUp Timer  
associated Exti line  
flag.

**Return value:**

- None.

\_\_HAL\_RTC\_WAKEUPTIMER\_EXTI\_GENERATE\_SWIT

**Description:**

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

**Return value:**

- None.

\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_ENABLE\_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

\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_DISABLE\_FALLING\_EDGE

trigger on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

**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.



\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_GET\_FLAG

**Return value:**

- None.

**Description:**

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

**Return value:**

- Line: Status.

\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_CLEAR\_FLAG

**Description:**

- Clear the RTC Tamper and Timestamp associated Exti line flag.

**Return value:**

- None.

\_\_HAL\_RTC\_TAMPER\_TIMESTAMP\_EXTI\_GENERATE\_SWIT

**Description:**

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

**Return value:**

- None.

***Private macros to check input parameters***

IS\_RTC\_OUTPUT

IS\_RTC\_BKP

IS\_TIMESTAMP\_EDGE

IS\_RTC\_TAMPER

IS\_RTC\_TAMPER\_INTERRUPT

IS\_RTC\_TIMESTAMP\_PIN

IS\_RTC\_TAMPER\_TRIGGER

IS\_RTC\_TAMPER\_ERASE\_MODE

IS\_RTC\_TAMPER\_MASKFLAG\_STATE

IS\_RTC\_TAMPER\_FILTER

IS\_RTC\_TAMPER\_SAMPLING\_FREQ

IS\_RTC\_TAMPER\_PRECHARGE\_DURATION

IS\_RTC\_TAMPER\_TIMESTAMPONTAMPER\_DETECTION

IS\_RTC\_TAMPER\_PULLUP\_STATE

IS\_RTC\_WAKEUP\_CLOCK  
 IS\_RTC\_WAKEUP\_COUNTER  
 IS\_RTC\_SMOOTH\_CALIB\_PERIOD  
 IS\_RTC\_SMOOTH\_CALIB\_PLUS  
 IS\_RTC\_SMOOTH\_CALIB\_MINUS  
 IS\_RTC\_SHIFT\_ADD1S  
 IS\_RTC\_SHIFT\_SUBFS  
 IS\_RTC\_CALIB\_OUTPUT

#### ***RTCEx Output selection Definitions***

RTC\_OUTPUT\_DISABLE  
 RTC\_OUTPUT\_ALARMA  
 RTC\_OUTPUT\_ALARMB  
 RTC\_OUTPUT\_WAKEUP

#### ***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

#### ***RTCEx Smooth calib period Definitions***

RTC_SMOOTHCALIB_PERIOD_32SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_16SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_8SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK seconds

#### ***RTCEx Smooth calib Plus pulses Definitions***

RTC_SMOOTHCALIB_PLUSPULSES_SET	The number of RTCCLK pulses added during a X -second window = $Y - \text{CALM}[8:0]$ with $Y = 512, 256, 128$ when $X = 32, 16, 8$
RTC_SMOOTHCALIB_PLUSPULSES_RESET	The number of RTCCLK pulses subbstited during a 32-second window = $\text{CALM}[8:0]$

#### ***RTCEx Tamper EraseBackUp Definitions***

RTC\_TAMPER\_ERASE\_BACKUP\_ENABLE  
 RTC\_TAMPER\_ERASE\_BACKUP\_DISABLE

#### ***RTCEx Tamper Filter Definitions***

RTC_TAMPERFILTER_DISABLE	Tamper filter is disabled
RTC_TAMPERFILTER_2SAMPLE	Tamper is activated after 2 consecutive samples at

the active level

RTC\_TAMPERFILTER\_4SAMPLE Tamper is activated after 4 consecutive samples at the active level

RTC\_TAMPERFILTER\_8SAMPLE Tamper is activated after 8 consecutive samples at the active level.

#### ***RTCEx Tamper Interrupt Definitions***

RTC\_TAMPER1\_INTERRUPT

RTC\_TAMPER2\_INTERRUPT

RTC\_TAMPER3\_INTERRUPT

RTC\_ALL\_TAMPER\_INTERRUPT

#### ***RTCEx Tamper MaskFlag Definitions***

RTC\_TAMPERMASK\_FLAG\_DISABLE

RTC\_TAMPERMASK\_FLAG\_ENABLE

#### ***RTCEx Tamper Pins Definitions***

RTC\_TAMPER\_1

RTC\_TAMPER\_2

RTC\_TAMPER\_3

#### ***RTCEx Tamper Pin Precharge Duration Definitions***

RTC\_TAMPERPRECHARGEDURATION\_1RTCCLK Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

RTC\_TAMPERPRECHARGEDURATION\_2RTCCLK Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

RTC\_TAMPERPRECHARGEDURATION\_4RTCCLK Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

RTC\_TAMPERPRECHARGEDURATION\_8RTCCLK Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

#### ***RTCEx Tamper Pull UP Definitions***

RTC\_TAMPER\_PULLUP\_ENABLE TimeStamp on Tamper Detection event saved

RTC\_TAMPER\_PULLUP\_DISABLE TimeStamp on Tamper Detection event is not saved

#### ***RTCEx Tamper Sampling Frequencies Definitions***

RTC\_TAMPERSAMPLINGFREQ\_RTCCLK\_DIV32768 Each of the tamper inputs are sampled with a frequency =  $\text{RTCCLK} / 32768$

RTC\_TAMPERSAMPLINGFREQ\_RTCCLK\_DIV16384 Each of the tamper inputs are sampled with a frequency =  $\text{RTCCLK} / 16384$

RTC\_TAMPERSAMPLINGFREQ\_RTCCLK\_DIV8192 Each of the tamper inputs are sampled with a frequency =  $\text{RTCCLK} / 8192$

	RTCCLK / 8192
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

***RTCEX Tamper TimeStampOnTamperDetection Definitions***

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TIMESTAMPONTAMPERDETECTION_DISABLE	TimeStamp on Tamper Detection event is not saved

***RTCEX Tamper Trigger Definitions***

RTC\_TAMPERTRIGGER\_RISINGEDGE  
 RTC\_TAMPERTRIGGER\_FALLINGEDGE  
 RTC\_TAMPERTRIGGER\_LOWLEVEL  
 RTC\_TAMPERTRIGGER\_HIGHLEVEL

***RTCEX TimeStamp Pin Selection***

RTC\_TIMESTAMPPIN\_DEFAULT  
 RTC\_TIMESTAMPPIN\_PI8  
 RTC\_TIMESTAMPPIN\_PC1

***RTCEX Time Stamp Edges definitions***

RTC\_TIMESTAMPEDGE\_RISING  
 RTC\_TIMESTAMPEDGE\_FALLING

***RTCEX Wakeup Timer Definitions***

RTC\_WAKEUPCLOCK\_RTCCLK\_DIV16  
 RTC\_WAKEUPCLOCK\_RTCCLK\_DIV8  
 RTC\_WAKEUPCLOCK\_RTCCLK\_DIV4  
 RTC\_WAKEUPCLOCK\_RTCCLK\_DIV2  
 RTC\_WAKEUPCLOCK\_CK\_SPRE\_16BITS  
 RTC\_WAKEUPCLOCK\_CK\_SPRE\_17BITS

## 48 HAL SAI Generic Driver

### 48.1 SAI Firmware driver registers structures

#### 48.1.1 SAI\_InitTypeDef

##### Data Fields

- *uint32\_t AudioMode*
- *uint32\_t Synchro*
- *uint32\_t SynchroExt*
- *uint32\_t OutputDrive*
- *uint32\_t NoDivider*
- *uint32\_t FIFOThreshold*
- *uint32\_t AudioFrequency*
- *uint32\_t Mckdiv*
- *uint32\_t MonoStereoMode*
- *uint32\_t CompandingMode*
- *uint32\_t TriState*
- *uint32\_t Protocol*
- *uint32\_t DataSize*
- *uint32\_t FirstBit*
- *uint32\_t ClockStrobing*

##### Field Documentation

- ***uint32\_t SAI\_InitTypeDef::AudioMode***  
Specifies the SAI Block audio Mode. This parameter can be a value of [SAI\\_Block\\_Mode](#)
- ***uint32\_t SAI\_InitTypeDef::Synchro***  
Specifies SAI Block synchronization This parameter can be a value of [SAI\\_Block\\_Synchronization](#)
- ***uint32\_t SAI\_InitTypeDef::SynchroExt***  
Specifies SAI Block synchronization, this setup is common for BLOCKA and BLOCKB This parameter can be a value of [SAI\\_Block\\_SyncExt](#)
- ***uint32\_t SAI\_InitTypeDef::OutputDrive***  
Specifies when SAI Block outputs are driven. This parameter can be a value of [SAI\\_Block\\_Output\\_Drive](#)  
**Note:**this value has to be set before enabling the audio block but after the audio block configuration.
- ***uint32\_t SAI\_InitTypeDef::NoDivider***  
Specifies whether master clock will be divided or not. This parameter can be a value of [SAI\\_Block\\_NoDivider](#)  
**Note:**If bit NODIV in the SAI\_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NODIV in the SAI\_xCR1 register is set, the frame length can take any of the values without constraint since the input clock of the audio block should be equal to the bit clock. There is no MCLK\_x clock which can be output.

- **`uint32_t SAI_InitTypeDef::FIFOThreshold`**  
Specifies SAI Block FIFO threshold. This parameter can be a value of [SAI\\_Block\\_Fifo\\_Threshold](#)
- **`uint32_t SAI_InitTypeDef::AudioFrequency`**  
Specifies the audio frequency sampling. This parameter can be a value of [SAI\\_Audio\\_Frequency](#)
- **`uint32_t SAI_InitTypeDef::Mckdiv`**  
Specifies the master clock divider, the parameter will be used if for AudioFrequency the user choice This parameter must be a number between Min\_Data = 0 and Max\_Data = 15
- **`uint32_t SAI_InitTypeDef::MonoStereoMode`**  
Specifies if the mono or stereo mode is selected. This parameter can be a value of [SAI\\_Mono\\_Stereo\\_Mode](#)
- **`uint32_t SAI_InitTypeDef::CompandingMode`**  
Specifies the companding mode type. This parameter can be a value of [SAI\\_Block\\_Companding\\_Mode](#)
- **`uint32_t SAI_InitTypeDef::TriState`**  
Specifies the companding mode type. This parameter can be a value of [SAI\\_TRISate\\_Management](#)
- **`uint32_t SAI_InitTypeDef::Protocol`**  
Specifies the SAI Block protocol. This parameter can be a value of [SAI\\_Block\\_Protocol](#)
- **`uint32_t SAI_InitTypeDef::DataSize`**  
Specifies the SAI Block data size. This parameter can be a value of [SAI\\_Block\\_Data\\_Size](#)
- **`uint32_t SAI_InitTypeDef::FirstBit`**  
Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [SAI\\_Block\\_MSB\\_LSB\\_transmission](#)
- **`uint32_t SAI_InitTypeDef::ClockStrobing`**  
Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of [SAI\\_Block\\_Clock\\_Strobing](#)

### 48.1.2 SAI\_FramelnitTypeDef

#### Data Fields

- **`uint32_t FrameLength`**
- **`uint32_t ActiveFrameLength`**
- **`uint32_t FSDefinition`**
- **`uint32_t FSPolarity`**
- **`uint32_t FSOOffset`**

#### Field Documentation

- **`uint32_t SAI_FramelnitTypeDef::FrameLength`**  
Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min\_Data = 8 and Max\_Data = 256.  
**Note:** If master clock MCLK\_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.
- **`uint32_t SAI_FramelnitTypeDef::ActiveFrameLength`**  
Specifies the Frame synchronization active level length. This Parameter specifies the

length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min\_Data = 1 and Max\_Data = 128

- ***uint32\_t SAI\_FramelnitTypeDef::FSDefinition***  
Specifies the Frame synchronization definition. This parameter can be a value of [SAI\\_Block\\_FS\\_Definition](#)
- ***uint32\_t SAI\_FramelnitTypeDef::FSPolarity***  
Specifies the Frame synchronization Polarity. This parameter can be a value of [SAI\\_Block\\_FS\\_Polarity](#)
- ***uint32\_t SAI\_FramelnitTypeDef::FSOffset***  
Specifies the Frame synchronization Offset. This parameter can be a value of [SAI\\_Block\\_FS\\_Offset](#)

### 48.1.3 SAI\_SlotInitTypeDef

#### Data Fields

- ***uint32\_t FirstBitOffset***
- ***uint32\_t SlotSize***
- ***uint32\_t SlotNumber***
- ***uint32\_t SlotActive***

#### Field Documentation

- ***uint32\_t SAI\_SlotInitTypeDef::FirstBitOffset***  
Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min\_Data = 0 and Max\_Data = 24
- ***uint32\_t SAI\_SlotInitTypeDef::SlotSize***  
Specifies the Slot Size. This parameter can be a value of [SAI\\_Block\\_Slot\\_Size](#)
- ***uint32\_t SAI\_SlotInitTypeDef::SlotNumber***  
Specifies the number of slot in the audio frame. This parameter must be a number between Min\_Data = 1 and Max\_Data = 16
- ***uint32\_t SAI\_SlotInitTypeDef::SlotActive***  
Specifies the slots in audio frame that will be activated. This parameter can be a value of [SAI\\_Block\\_Slot\\_Active](#)

### 48.1.4 \_\_SAI\_HandleTypeDef

#### Data Fields

- ***SAI\_Block\_TypeDef \* Instance***
- ***SAI\_InitTypeDef Init***
- ***SAI\_FramelnitTypeDef Framelnit***
- ***SAI\_SlotInitTypeDef SlotInit***
- ***uint8\_t \* pBuffPtr***
- ***uint16\_t XferSize***
- ***uint16\_t XferCount***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***SAIcallback mutecallback***

- ***void(\* InterruptServiceRoutine***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_SAI\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***SAI\_BlockTypeDef \_\_SAI\_HandleTypeDef::Instance***  
SAI Blockx registers base address
- ***SAI\_InitTypeDef \_\_SAI\_HandleTypeDef::Init***  
SAI communication parameters
- ***SAI\_FrameInitTypeDef \_\_SAI\_HandleTypeDef::FrameInit***  
SAI Frame configuration parameters
- ***SAI\_SlotInitTypeDef \_\_SAI\_HandleTypeDef::SlotInit***  
SAI Slot configuration parameters
- ***uint8\_t\* \_\_SAI\_HandleTypeDef::pBuffPtr***  
Pointer to SAI transfer Buffer
- ***uint16\_t \_\_SAI\_HandleTypeDef::XferSize***  
SAI transfer size
- ***uint16\_t \_\_SAI\_HandleTypeDef::XferCount***  
SAI transfer counter
- ***DMA\_HandleTypeDef \_\_SAI\_HandleTypeDef::hdmatx***  
SAI Tx DMA handle parameters
- ***DMA\_HandleTypeDef \_\_SAI\_HandleTypeDef::hdmarx***  
SAI Rx DMA handle parameters
- ***SAIcallback \_\_SAI\_HandleTypeDef::mutecallback***  
SAI mute callback
- ***void(\* \_\_SAI\_HandleTypeDef::InterruptServiceRoutine)(struct \_\_SAI\_HandleTypeDef \*hsai)***
- ***HAL\_LockTypeDef \_\_SAI\_HandleTypeDef::Lock***  
SAI locking object
- ***\_\_IO HAL\_SAI\_StateTypeDef \_\_SAI\_HandleTypeDef::State***  
SAI communication state
- ***\_\_IO uint32\_t \_\_SAI\_HandleTypeDef::ErrorCode***  
SAI Error code

## 48.2 SAI Firmware driver API description

### 48.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a `SAI_HandleTypeDef` handle structure.
2. Initialize the SAI low level resources by implementing the `HAL_SAI_MspInit()` API:
  - a. Enable the SAI interface clock.
  - b. SAI pins configuration:
    - Enable the clock for the SAI GPIOs.
    - Configure these SAI pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (`HAL_SAI_Transmit_IT()` and `HAL_SAI_Receive_IT()` APIs):
    - Configure the SAI interrupt priority.
    - Enable the NVIC SAI IRQ handle.



- d. DMA Configuration if you need to use DMA process (HAL\_SAI\_Transmit\_DMA() and HAL\_SAI\_Receive\_DMA() APIs):
  - Declare a DMA handle structure for the Tx/Rx stream.
  - Enable the DMAx interface clock.
  - Configure the declared DMA handle structure with the required Tx/Rx parameters.
  - Configure the DMA Tx/Rx Stream.
  - Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
  - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. Program the SAI Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL\_SAI\_Init() function. The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros \_\_SAI\_ENABLE\_IT() and \_\_SAI\_DISABLE\_IT() inside the transmit and receive process.



SAI Clock Source, the configuration is managed through RCCEx\_PeriphCLKConfig() function in the HAL RCC drivers



Make sure that either:

- I2S PLL is configured or
- SAI PLL is configured or
- External clock source is configured after setting correctly the define constant EXTERNAL\_CLOCK\_VALUE in the stm32f7xx\_hal\_conf.h file.



In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.



In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.



It is mandatory to respect the following conditions in order to avoid bad SAI behavior:

- First bit Offset  $\leq$  (SLOT size - Data size)
- Data size  $\leq$  SLOT size
- Number of SLOT x SLOT size = Frame length
- The number of slots should be even when SAI\_FS\_CHANNEL\_IDENTIFICATION is selected.

Three operation modes are available within this driver :

### Polling mode IO operation

- Send an amount of data in blocking mode using HAL\_SAI\_Transmit()

- Receive an amount of data in blocking mode using HAL\_SAI\_Receive()

### Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL\_SAI\_Transmit\_IT()
- At transmission end of transfer HAL\_SAI\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SAI\_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL\_SAI\_Receive\_IT()
- At reception end of transfer HAL\_SAI\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SAI\_RxCpltCallback
- In case of transfer Error, HAL\_SAI\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SAI\_ErrorCallback

### DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL\_SAI\_Transmit\_DMA()
- At transmission end of transfer HAL\_SAI\_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SAI\_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL\_SAI\_Receive\_DMA()
- At reception end of transfer HAL\_SAI\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SAI\_RxCpltCallback
- In case of transfer Error, HAL\_SAI\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SAI\_ErrorCallback
- Pause the DMA Transfer using HAL\_SAI\_DMABase()
- Resume the DMA Transfer using HAL\_SAI\_DMAResume()
- Stop the DMA Transfer using HAL\_SAI\_DMAStop()

### SAI HAL driver macros list

Below the list of most used macros in USART HAL driver :

- \_\_HAL\_SAI\_ENABLE: Enable the SAI peripheral
- \_\_HAL\_SAI\_DISABLE: Disable the SAI peripheral
- \_\_HAL\_SAI\_ENABLE\_IT : Enable the specified SAI interrupts
- \_\_HAL\_SAI\_DISABLE\_IT : Disable the specified SAI interrupts
- \_\_HAL\_SAI\_GET\_IT\_SOURCE: Check if the specified SAI interrupt source is enabled or disabled
- \_\_HAL\_SAI\_GET\_FLAG: Check whether the specified SAI flag is set or not

## 48.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SAIx peripheral:

- User must implement HAL\_SAI\_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function HAL\_SAI\_Init() to configure the selected device with the selected configuration:
  - Mode (Master/slave TX/RX)
  - Protocol
  - Data Size

- MCLK Output
- Audio frequency
- FIFO Threshold
- Frame Config
- Slot Config
- Call the function HAL\_SAI\_DeInit() to restore the default configuration of the selected SAI peripheral.

This section contains the following APIs:

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

### 48.2.3 IO operation functions

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

- There are two modes of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SAI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- Blocking mode functions are :
  - HAL\_SAI\_Transmit()
  - HAL\_SAI\_Receive()
  - HAL\_SAI\_TransmitReceive()
- Non Blocking mode functions with Interrupt are :
  - HAL\_SAI\_Transmit\_IT()
  - HAL\_SAI\_Receive\_IT()
  - HAL\_SAI\_TransmitReceive\_IT()
- Non Blocking mode functions with DMA are :
  - HAL\_SAI\_Transmit\_DMA()
  - HAL\_SAI\_Receive\_DMA()
  - HAL\_SAI\_TransmitReceive\_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL\_SAI\_TxCpltCallback()
  - HAL\_SAI\_RxCpltCallback()
  - HAL\_SAI\_ErrorCallback()

This section contains the following APIs:

- [\*HAL\\_SAI\\_Transmit\(\)\*](#)
- [\*HAL\\_SAI\\_Receive\(\)\*](#)
- [\*HAL\\_SAI\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_SAI\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_SAI\\_DMAPause\(\)\*](#)
- [\*HAL\\_SAI\\_DMAResume\(\)\*](#)
- [\*HAL\\_SAI\\_DMAStop\(\)\*](#)
- [\*HAL\\_SAI\\_Abort\(\)\*](#)
- [\*HAL\\_SAI\\_Transmit\\_DMA\(\)\*](#)

- [HAL\\_SAI\\_Receive\\_DMA\(\)](#)
- [HAL\\_SAI\\_EnableTxMuteMode\(\)](#)
- [HAL\\_SAI\\_DisableTxMuteMode\(\)](#)
- [HAL\\_SAI\\_EnableRxMuteMode\(\)](#)
- [HAL\\_SAI\\_DisableRxMuteMode\(\)](#)
- [HAL\\_SAI\\_IRQHandler\(\)](#)
- [HAL\\_SAI\\_TxCpltCallback\(\)](#)
- [HAL\\_SAI\\_TxHalfCpltCallback\(\)](#)
- [HAL\\_SAI\\_RxCpltCallback\(\)](#)
- [HAL\\_SAI\\_RxHalfCpltCallback\(\)](#)
- [HAL\\_SAI\\_ErrorCallback\(\)](#)

#### 48.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- [HAL\\_SAI\\_GetState\(\)](#)
- [HAL\\_SAI\\_GetError\(\)](#)

#### 48.2.5 HAL\_SAI\_InitProtocol

Function Name	<b>HAL_StatusTypeDef HAL_SAI_InitProtocol</b> (SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)
Function Description	Initializes the structure FrameInit, SlotInit and the low part of Init according to the specified parameters and call the function HAL_SAI_Init to initialize the SAI block.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b>: : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>protocol</b>: : one of the supported protocol SAI Supported protocol</li> <li>• <b>datasize</b>: : one of the supported datasize SAI protocol data size the configuration information for SAI module.</li> <li>• <b>nbslot</b>: : Number of slot.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.6 HAL\_SAI\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Init</b> (SAI_HandleTypeDef * hsai)
Function Description	Initializes the SAI according to the specified parameters in the SAI_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b>: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.7 HAL\_SAI\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DeInit</b> (SAI_HandleTypeDef * hsai)
Function Description	DeInitializes the SAI peripheral.

Parameters	<ul style="list-style-type: none"> <li>• <b>hsai:</b> pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.8 HAL\_SAI\_Msplnit

Function Name	<b>void HAL_SAI_Msplnit (SAI_HandleTypeDef * hsai)</b>
Function Description	SAI MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai:</b> pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 48.2.9 HAL\_SAI\_MspDeInit

Function Name	<b>void HAL_SAI_MspDeInit (SAI_HandleTypeDef * hsai)</b>
Function Description	SAI MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai:</b> pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 48.2.10 HAL\_SAI\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmits an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai:</b> pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be sent</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.11 HAL\_SAI\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receives an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai:</b> pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData:</b> Pointer to data buffer</li> <li>• <b>Size:</b> Amount of data to be received</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.12 HAL\_SAI\_Transmit\_IT

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

---

**(SAI\_HandleTypeDef \* hsai, uint8\_t \* pData, uint16\_t Size)**

Function Description	Transmits an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b>: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.13 HAL\_SAI\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Receive_IT</b> <b>(SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)</b>
Function Description	Receives an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b>: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li>• <b>pData</b>: Pointer to data buffer</li> <li>• <b>Size</b>: Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.14 HAL\_SAI\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DMAPause</b> <b>(SAI_HandleTypeDef * hsai)</b>
Function Description	Pauses the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b>: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.15 HAL\_SAI\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DMAResume</b> <b>(SAI_HandleTypeDef * hsai)</b>
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b>: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 48.2.16 HAL\_SAI\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DMAStop</b> <b>(SAI_HandleTypeDef * hsai)</b>
Function Description	Stops the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsai</b>: : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**48.2.17 HAL\_SAI\_Abort**

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)</b>
Function Description	Abort the current transfer and disabled the SAI.
Parameters	<ul style="list-style-type: none"> <li><b>hsai</b>: : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**48.2.18 HAL\_SAI\_Transmit\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)</b>
Function Description	Transmits an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hsai</b>: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li><b>pData</b>: Pointer to data buffer</li> <li><b>Size</b>: Amount of data to be sent</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**48.2.19 HAL\_SAI\_Receive\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)</b>
Function Description	Receives an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li><b>hsai</b>: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li><b>pData</b>: Pointer to data buffer</li> <li><b>Size</b>: Amount of data to be received</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**48.2.20 HAL\_SAI\_EnableTxMuteMode**

Function Name	<b>HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)</b>
Function Description	Enable the tx mute mode.
Parameters	<ul style="list-style-type: none"> <li><b>hsai</b>: : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.</li> <li><b>val</b>: : value sent during the mute SAI Block Mute Value</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**48.2.21 HAL\_SAI\_DisableTxMuteMode**

Function Name	<b>HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)</b>
Function Description	Disable the tx mute mode.
Parameters	<ul style="list-style-type: none"> <li><b>hsai</b>: : pointer to a SAI_HandleTypeDef structure that</li> </ul>

contains the configuration information for SAI module.

Return values

- HAL status

#### 48.2.22 HAL\_SAI\_EnableRxMuteMode

Function Name **HAL\_StatusTypeDef HAL\_SAI\_EnableRxMuteMode (SAI\_HandleTypeDef \* hsai, SAIcallback callback, uint16\_t counter)**

Function Description Enable the rx mute detection.

Parameters

- **hsai**: : pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.
- **callback**: : function called when the mute is detected
- **counter**: : number a data before mute detection max 63.

Return values

- HAL status

#### 48.2.23 HAL\_SAI\_DisableRxMuteMode

Function Name **HAL\_StatusTypeDef HAL\_SAI\_DisableRxMuteMode (SAI\_HandleTypeDef \* hsai)**

Function Description Disable the rx mute detection.

Parameters

- **hsai**: : pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- HAL status

#### 48.2.24 HAL\_SAI\_IRQHandler

Function Name **void HAL\_SAI\_IRQHandler (SAI\_HandleTypeDef \* hsai)**

Function Description This function handles SAI interrupt request.

Parameters

- **hsai**: pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- HAL status

#### 48.2.25 HAL\_SAI\_TxCpltCallback

Function Name **void HAL\_SAI\_TxCpltCallback (SAI\_HandleTypeDef \* hsai)**

Function Description Tx Transfer completed callbacks.

Parameters

- **hsai**: pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- None

#### 48.2.26 HAL\_SAI\_TxHalfCpltCallback

Function Name **void HAL\_SAI\_TxHalfCpltCallback (SAI\_HandleTypeDef \* hsai)**

Function Description Tx Transfer Half completed callbacks.

Parameters

- **hsai**: pointer to a SAI\_HandleTypeDef structure that contains



the configuration information for SAI module.

Return values

- None

#### 48.2.27 HAL\_SAI\_RxCpltCallback

Function Name **void HAL\_SAI\_RxCpltCallback (SAI\_HandleTypeDef \* hsai)**

Function Description Rx Transfer completed callbacks.

Parameters

- **hsai:** pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- None

#### 48.2.28 HAL\_SAI\_RxHalfCpltCallback

Function Name **void HAL\_SAI\_RxHalfCpltCallback (SAI\_HandleTypeDef \* hsai)**

Function Description Rx Transfer half completed callbacks.

Parameters

- **hsai:** pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- None

#### 48.2.29 HAL\_SAI\_ErrorCallback

Function Name **void HAL\_SAI\_ErrorCallback (SAI\_HandleTypeDef \* hsai)**

Function Description SAI error callbacks.

Parameters

- **hsai:** pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- None

#### 48.2.30 HAL\_SAI\_GetState

Function Name **HAL\_SAI\_StateTypeDef HAL\_SAI\_GetState (SAI\_HandleTypeDef \* hsai)**

Function Description Returns the SAI state.

Parameters

- **hsai:** pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- HAL state

#### 48.2.31 HAL\_SAI\_GetError

Function Name **uint32\_t HAL\_SAI\_GetError (SAI\_HandleTypeDef \* hsai)**

Function Description Return the SAI error code.

Parameters

- **hsai:** : pointer to a SAI\_HandleTypeDef structure that contains the configuration information for the specified SAI Block.

Return values

- SAI Error Code

## 48.3 SAI Firmware driver defines

### 48.3.1 SAI

#### ***SAI Audio Frequency***

SAI\_AUDIO\_FREQUENCY\_192K

SAI\_AUDIO\_FREQUENCY\_96K

SAI\_AUDIO\_FREQUENCY\_48K

SAI\_AUDIO\_FREQUENCY\_44K

SAI\_AUDIO\_FREQUENCY\_32K

SAI\_AUDIO\_FREQUENCY\_22K

SAI\_AUDIO\_FREQUENCY\_16K

SAI\_AUDIO\_FREQUENCY\_11K

SAI\_AUDIO\_FREQUENCY\_8K

SAI\_AUDIO\_FREQUENCY\_MCKDIV

#### ***SAI Block Clock Strobing***

SAI\_CLOCKSTROBING\_FALLINGEDGE

SAI\_CLOCKSTROBING\_RISINGEDGE

#### ***SAI Block Companding Mode***

SAI\_NOCOMPANDING

SAI\_ULAW\_1CPL\_COMPANDING

SAI\_ALAW\_1CPL\_COMPANDING

SAI\_ULAW\_2CPL\_COMPANDING

SAI\_ALAW\_2CPL\_COMPANDING

#### ***SAI Block Data Size***

SAI\_DATASIZE\_8

SAI\_DATASIZE\_10

SAI\_DATASIZE\_16

SAI\_DATASIZE\_20

SAI\_DATASIZE\_24

SAI\_DATASIZE\_32

#### ***SAI Block Fifo Status Level***

SAI\_FIFOSTATUS\_EMPTY

SAI\_FIFOSTATUS\_LESS1QUARTERFULL

SAI\_FIFOSTATUS\_1QUARTERFULL

SAI\_FIFOSTATUS\_HALFFULL

SAI\_FIFOSTATUS\_3QUARTERFULL

SAI\_FIFOSTATUS\_FULL

**SAI Block Fifo Threshold**

SAI\_FIFOTHRESHOLD\_EMPTY

SAI\_FIFOTHRESHOLD\_1QF

SAI\_FIFOTHRESHOLD\_HF

SAI\_FIFOTHRESHOLD\_3QF

SAI\_FIFOTHRESHOLD\_FULL

**SAI Block Flags Definition**

SAI\_FLAG\_OVRUDR

SAI\_FLAG\_MUTEDDET

SAI\_FLAG\_WCKCFG

SAI\_FLAG\_FREQ

SAI\_FLAG\_CNRDY

SAI\_FLAG\_AFSDET

SAI\_FLAG\_LFSDET

**SAI Block FS Definition**

SAI\_FS\_STARTFRAME

SAI\_FS\_CHANNEL\_IDENTIFICATION

**SAI Block FS Offset**

SAI\_FS\_FIRSTBIT

SAI\_FS\_BEFOREFIRSTBIT

**SAI Block FS Polarity**

SAI\_FS\_ACTIVE\_LOW

SAI\_FS\_ACTIVE\_HIGH

**SAI Block Interrupts Definition**

SAI\_IT\_OVRUDR

SAI\_IT\_MUTEDDET

SAI\_IT\_WCKCFG

SAI\_IT\_FREQ

SAI\_IT\_CNRDY

SAI\_IT\_AFSDET

SAI\_IT\_LFSDET

**SAI Block Mode**

SAI\_MODEMASTER\_TX

SAI\_MODEMASTER\_RX

SAI\_MODESLAVE\_TX

SAI\_MODESLAVE\_RX

**SAI Block MSB LSB transmission**

SAI\_FIRSTBIT\_MSB

SAI\_FIRSTBIT\_LSB

**SAI Block Mute Value**

SAI\_ZERO\_VALUE

SAI\_LAST\_SENT\_VALUE

**SAI Block NoDivider**

SAI\_MASTERDIVIDER\_ENABLE

SAI\_MASTERDIVIDER\_DISABLE

**SAI Block Output Drive**

SAI\_OUTPUTDRIVE\_DISABLE

SAI\_OUTPUTDRIVE\_ENABLE

**SAI Block Protocol**

SAI\_FREE\_PROTOCOL

SAI\_SPDIF\_PROTOCOL

SAI\_AC97\_PROTOCOL

**SAI Block Slot Active**

SAI\_SLOT\_NOTACTIVE

SAI\_SLOTACTIVE\_0

SAI\_SLOTACTIVE\_1

SAI\_SLOTACTIVE\_2

SAI\_SLOTACTIVE\_3

SAI\_SLOTACTIVE\_4

SAI\_SLOTACTIVE\_5

SAI\_SLOTACTIVE\_6

SAI\_SLOTACTIVE\_7

SAI\_SLOTACTIVE\_8

SAI\_SLOTACTIVE\_9

SAI\_SLOTACTIVE\_10

SAI\_SLOTACTIVE\_11

SAI\_SLOTACTIVE\_12

SAI\_SLOTACTIVE\_13

SAI\_SLOTACTIVE\_14

SAI\_SLOTACTIVE\_15

SAI\_SLOTACTIVE\_ALL

**SAI Block Slot Size**

SAI\_SLOTSIZE\_DATASIZE

SAI\_SLOTSIZE\_16B

SAI\_SLOTSIZE\_32B

**SAI External synchronisation**

SAI\_SYNCEXT\_DISABLE

SAI\_SYNCEXT\_IN\_ENABLE

SAI\_SYNCEXT\_OUTBLOCKA\_ENABLE

SAI\_SYNCEXT\_OUTBLOCKB\_ENABLE

**SAI Block Synchronization**

SAI\_ASYNCHRONOUS

SAI\_SYNCHRONOUS

SAI\_SYNCHRONOUS\_EXT

**SAI Clock Source**

SAI\_CLKSOURCE\_PLLSAI

SAI\_CLKSOURCE\_PLLI2S

SAI\_CLKSOURCE\_EXT

SAI\_CLKSOURCE\_NA

**SAI Error Code**

HAL_SAI_ERROR_NONE	No error
HAL_SAI_ERROR_OVR	Overrun Error
HAL_SAI_ERROR_UDR	Underrun error
HAL_SAI_ERROR_AFSDET	Anticipated Frame synchronisation detection
HAL_SAI_ERROR_LFSDET	Late Frame synchronisation detection
HAL_SAI_ERROR_CNREADY	codec not ready
HAL_SAI_ERROR_WCKCFG	Wrong clock configuration
HAL_SAI_ERROR_TIMEOUT	Timeout error

**SAI Exported Macros****\_\_HAL\_SAI\_RESET\_HANDLE\_STATE**    **Description:**

- Reset SAI handle state.

**Parameters:**

- `__HANDLE__`: specifies the SAI Handle.

**Return value:**

- None

**\_\_HAL\_SAI\_ENABLE\_IT****Description:**

- Enable or disable the specified SAI interrupts.

**Parameters:**

- `__HANDLE__`: specifies the SAI Handle.

- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - `SAI_IT_OVRUDR`: Overrun underrun interrupt enable
  - `SAI_IT_MUTEDET`: Mute detection interrupt enable
  - `SAI_IT_WCKCFG`: Wrong Clock Configuration interrupt enable
  - `SAI_IT_FREQ`: FIFO request interrupt enable
  - `SAI_IT_CNRDY`: Codec not ready interrupt enable
  - `SAI_IT_AFSDET`: Anticipated frame synchronization detection interrupt enable
  - `SAI_IT_LFSDET`: Late frame synchronization detection interrupt enable

**Return value:**

- None

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SAI Handle. This parameter can be SAI where x: 1, 2, or 3 to select the SAI peripheral.
- `__INTERRUPT__`: specifies the SAI interrupt source to check. This parameter can be one of the following values:
  - `SAI_IT_TXE`: Tx buffer empty interrupt enable.
  - `SAI_IT_RXNE`: Rx buffer not empty interrupt enable.
  - `SAI_IT_ERR`: Error interrupt enable.

**Return value:**

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

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SAI Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:

`__HAL_SAI_DISABLE_IT``__HAL_SAI_GET_IT_SOURCE``__HAL_SAI_GET_FLAG`

- SAI\_FLAG\_OVRUDR: Overrun underrun flag.
- SAI\_FLAG\_MUTEDET: Mute detection flag.
- SAI\_FLAG\_WCKCFG: Wrong Clock Configuration flag.
- SAI\_FLAG\_FREQ: FIFO request flag.
- SAI\_FLAG\_CNRDY: Codec not ready flag.
- SAI\_FLAG\_AFSDET: Anticipated frame synchronization detection flag.
- SAI\_FLAG\_LFSDET: Late frame synchronization detection flag.

**Return value:**

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

**Description:**

- Clears the specified SAI pending flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SAI Handle.
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be any combination of the following values:
  - SAI\_FLAG\_OVRUDR: Clear Overrun underrun
  - SAI\_FLAG\_MUTEDET: Clear Mute detection
  - SAI\_FLAG\_WCKCFG: Clear Wrong Clock Configuration
  - SAI\_FLAG\_FREQ: Clear FIFO request
  - SAI\_FLAG\_CNRDY: Clear Codec not ready
  - SAI\_FLAG\_AFSDET: Clear Anticipated frame synchronization detection
  - SAI\_FLAG\_LFSDET: Clear Late frame synchronization detection

**Return value:**

- None

\_\_HAL\_SAI\_CLEAR\_FLAG

\_\_HAL\_SAI\_ENABLE

\_\_HAL\_SAI\_DISABLE

**SAI Mono Stereo Mode**

SAI\_STEREOMODE

SAI\_MONOMODE

**SAI Private Constants**

SAI\_FIFO\_SIZE

SAI\_DEFAULT\_TIMEOUT

***SAI\_Private\_Macros***

IS\_SAI\_BLOCK\_SYNCEXT  
IS\_SAI\_SUPPORTED\_PROTOCOL  
IS\_SAI\_PROTOCOL\_DATASIZE  
IS\_SAI\_CLK\_SOURCE  
IS\_SAI\_AUDIO\_FREQUENCY  
IS\_SAI\_BLOCK\_MODE  
IS\_SAI\_BLOCK\_PROTOCOL  
IS\_SAI\_BLOCK\_DATASIZE  
IS\_SAI\_BLOCK\_FIRST\_BIT  
IS\_SAI\_BLOCK\_CLOCK\_STROBING  
IS\_SAI\_BLOCK\_SYNCHRO  
IS\_SAI\_BLOCK\_OUTPUT\_DRIVE  
IS\_SAI\_BLOCK\_NODIVIDER  
IS\_SAI\_BLOCK\_FIFO\_STATUS  
IS\_SAI\_BLOCK\_MUTE\_COUNTER  
IS\_SAI\_BLOCK\_MUTE\_VALUE  
IS\_SAI\_BLOCK\_COMPANDING\_MODE  
IS\_SAI\_BLOCK\_FIFO\_THRESHOLD  
IS\_SAI\_BLOCK\_TRISTATE\_MANAGEMENT  
IS\_SAI\_MONO\_STEREO\_MODE  
IS\_SAI\_SLOT\_ACTIVE  
IS\_SAI\_BLOCK\_SLOT\_NUMBER  
IS\_SAI\_BLOCK\_SLOT\_SIZE  
IS\_SAI\_BLOCK\_FIRSTBIT\_OFFSET  
IS\_SAI\_BLOCK\_FS\_OFFSET  
IS\_SAI\_BLOCK\_FS\_POLARITY  
IS\_SAI\_BLOCK\_FS\_DEFINITION  
IS\_SAI\_BLOCK\_MASTER\_DIVIDER  
IS\_SAI\_BLOCK\_FRAME\_LENGTH  
IS\_SAI\_BLOCK\_ACTIVE\_FRAME

***SAI Supported protocol***

SAI\_I2S\_STANDARD  
SAI\_I2S\_MSBJUSTIFIED  
SAI\_I2S\_LSBJUSTIFIED  
SAI\_PCM\_LONG



SAI\_PCM\_SHORT

***SAI protocol data size***

SAI\_PROTOCOL\_DATASIZE\_16BIT

SAI\_PROTOCOL\_DATASIZE\_16BITEXTENDED

SAI\_PROTOCOL\_DATASIZE\_24BIT

SAI\_PROTOCOL\_DATASIZE\_32BIT

***SAI TRISate Management***

SAI\_OUTPUT\_NOTRELEASED

SAI\_OUTPUT\_RELEASED

## 49 HAL SAI Extension Driver

### 49.1 SAIEx Firmware driver API description

#### 49.1.1 SAI peripheral extension features

#### 49.1.2 How to use this driver

This driver provides functions to manage several sources to clock SAI

#### 49.1.3 Extension features Functions

This subsection provides a set of functions allowing to manage the possible SAI clock sources.

This section contains the following APIs:

- [SAI\\_BlockSynchroConfig\(\)](#)
- [SAI\\_GetInputClock\(\)](#)

#### 49.1.4 SAI\_BlockSynchroConfig

Function Name      **void SAI\_BlockSynchroConfig (SAI\_HandleTypeDef \* hsai)**

Function Description    Configure SAI Block synchronization mode.

Parameters              • **hsai:** pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values            • SAI Clock Input

#### 49.1.5 SAI\_GetInputClock

Function Name      **uint32\_t SAI\_GetInputClock (SAI\_HandleTypeDef \* hsai)**

Function Description    Get SAI Input Clock based on SAI source clock selection.

Parameters              • **hsai:** pointer to a SAI\_HandleTypeDef structure that contains the configuration information for SAI module.

Return values            • SAI Clock Input

## 50 HAL SDRAM Generic Driver

### 50.1 SDRAM Firmware driver registers structures

#### 50.1.1 SDRAM\_HandleTypeDef

##### Data Fields

- *FMC\_SDRAM\_TypeDef \* Instance*
- *FMC\_SDRAM\_InitTypeDef Init*
- *\_\_IO HAL\_SDRAM\_StateTypeDef State*
- *HAL\_LockTypeDef Lock*
- *DMA\_HandleTypeDef \* hdma*

##### Field Documentation

- *FMC\_SDRAM\_TypeDef\* SDRAM\_HandleTypeDef::Instance*  
Register base address
- *FMC\_SDRAM\_InitTypeDef SDRAM\_HandleTypeDef::Init*  
SDRAM device configuration parameters
- *\_\_IO HAL\_SDRAM\_StateTypeDef SDRAM\_HandleTypeDef::State*  
SDRAM access state
- *HAL\_LockTypeDef SDRAM\_HandleTypeDef::Lock*  
SDRAM locking object
- *DMA\_HandleTypeDef\* SDRAM\_HandleTypeDef::hdma*  
Pointer DMA handler

### 50.2 SDRAM Firmware driver API description

#### 50.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SDRAM memories. It uses the FMC layer functions to interface with SDRAM devices. The following sequence should be followed to configure the FMC to interface with SDRAM memories:

1. Declare a SDRAM\_HandleTypeDef handle structure, for example:  
SDRAM\_HandleTypeDef hdsram
  - Fill the SDRAM\_HandleTypeDef handle "Init" field with the allowed values of the structure member.
  - Fill the SDRAM\_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SDRAM device
2. Declare a FMC\_SDRAM\_TimingTypeDef structure; for example:  
FMC\_SDRAM\_TimingTypeDef Timing; and fill its fields with the allowed values of the structure member.
3. Initialize the SDRAM Controller by calling the function HAL\_SDRAM\_Init(). This function performs the following sequence:
  - a. MSP hardware layer configuration using the function HAL\_SDRAM\_MspInit()
  - b. Control register configuration using the FMC SDRAM interface function FMC\_SDRAM\_Init()

- c. Timing register configuration using the FMC SDRAM interface function `FMC_SDRAM_Timing_Init()`
  - d. Program the SDRAM external device by applying its initialization sequence according to the device plugged in your hardware. This step is mandatory for accessing the SDRAM device.
4. At this stage you can perform read/write accesses from/to the memory connected to the SDRAM Bank. You can perform either polling or DMA transfer using the following APIs:
  - `HAL_SDRAM_Read()/HAL_SDRAM_Write()` for polling read/write access
  - `HAL_SDRAM_Read_DMA()/HAL_SDRAM_Write_DMA()` for DMA read/write transfer
5. You can also control the SDRAM device by calling the control APIs `HAL_SDRAM_WriteOperation_Enable()/ HAL_SDRAM_WriteOperation_Disable()` to respectively enable/disable the SDRAM write operation or the function `HAL_SDRAM_SendCommand()` to send a specified command to the SDRAM device. The command to be sent must be configured with the `FMC_SDRAM_CommandTypeDef` structure.
6. You can continuously monitor the SDRAM device HAL state by calling the function `HAL_SDRAM_GetState()`

### 50.2.2 SDRAM Initialization and de\_initialization functions

This section provides functions allowing to initialize/de-initialize the SDRAM memory

This section contains the following APIs:

- [\*HAL\\_SDRAM\\_Init\(\)\*](#)
- [\*HAL\\_SDRAM\\_DeInit\(\)\*](#)
- [\*HAL\\_SDRAM\\_MspInit\(\)\*](#)
- [\*HAL\\_SDRAM\\_MspDeInit\(\)\*](#)
- [\*HAL\\_SDRAM\\_IRQHandler\(\)\*](#)
- [\*HAL\\_SDRAM\\_RefreshErrorCallback\(\)\*](#)
- [\*HAL\\_SDRAM\\_DMA\\_XferCpltCallback\(\)\*](#)
- [\*HAL\\_SDRAM\\_DMA\\_XferErrorCallback\(\)\*](#)

### 50.2.3 SDRAM Input and Output functions

This section provides functions allowing to use and control the SDRAM memory

This section contains the following APIs:

- [\*HAL\\_SDRAM\\_Read\\_8b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Write\\_8b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Read\\_16b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Write\\_16b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Read\\_32b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Write\\_32b\(\)\*](#)
- [\*HAL\\_SDRAM\\_Read\\_DMA\(\)\*](#)
- [\*HAL\\_SDRAM\\_Write\\_DMA\(\)\*](#)

### 50.2.4 SDRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SDRAM interface.

This section contains the following APIs:

- [\*HAL\\_SDRAM\\_WriteProtection\\_Enable\(\)\*](#)

- [HAL\\_SDRAM\\_WriteProtection\\_Disable\(\)](#)
- [HAL\\_SDRAM\\_SendCommand\(\)](#)
- [HAL\\_SDRAM\\_ProgramRefreshRate\(\)](#)
- [HAL\\_SDRAM\\_SetAutoRefreshNumber\(\)](#)
- [HAL\\_SDRAM\\_GetModeStatus\(\)](#)

### 50.2.5 SDRAM State functions

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

This section contains the following APIs:

- [HAL\\_SDRAM\\_GetState\(\)](#)

### 50.2.6 HAL\_SDRAM\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_Init</b> (SDRAM_HandleTypeDef * hsdram, FMC_SDRAM_TimingTypeDef * Timing)
Function Description	Performs the SDRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b>: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li>• <b>Timing</b>: Pointer to SDRAM control timing structure</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 50.2.7 HAL\_SDRAM\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SDRAM_DeInit</b> (SDRAM_HandleTypeDef * hsdram)
Function Description	Perform the SDRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b>: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 50.2.8 HAL\_SDRAM\_Msplnit

Function Name	<b>void HAL_SDRAM_Msplnit</b> (SDRAM_HandleTypeDef * hsdram)
Function Description	SDRAM MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b>: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 50.2.9 HAL\_SDRAM\_MspDeInit

Function Name	<b>void HAL_SDRAM_MspDeInit</b> (SDRAM_HandleTypeDef * hsdram)
Function Description	SDRAM MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b>: pointer to a SDRAM_HandleTypeDef structure that</li> </ul>

contains the configuration information for SDRAM module.

Return values • None

### 50.2.10 HAL\_SDRAM\_IRQHandler

Function Name **void HAL\_SDRAM\_IRQHandler (SDRAM\_HandleTypeDef \* hsdram)**

Function Description This function handles SDRAM refresh error interrupt request.

Parameters • **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values • HAL status

### 50.2.11 HAL\_SDRAM\_RefreshErrorCallback

Function Name **void HAL\_SDRAM\_RefreshErrorCallback (SDRAM\_HandleTypeDef \* hsdram)**

Function Description SDRAM Refresh error callback.

Parameters • **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values • None

### 50.2.12 HAL\_SDRAM\_DMA\_XferCpltCallback

Function Name **void HAL\_SDRAM\_DMA\_XferCpltCallback (DMA\_HandleTypeDef \* hdma)**

Function Description DMA transfer complete callback.

Parameters • **hdma**: pointer to a DMA\_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values • None

### 50.2.13 HAL\_SDRAM\_DMA\_XferErrorCallback

Function Name **void HAL\_SDRAM\_DMA\_XferErrorCallback (DMA\_HandleTypeDef \* hdma)**

Function Description DMA transfer complete error callback.

Parameters • **hdma**: DMA handle

Return values • None

### 50.2.14 HAL\_SDRAM\_Read\_8b

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_Read\_8b (SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress, uint8\_t \* pDstBuffer, uint32\_t BufferSize)**

Function Description Reads 8-bit data buffer from the SDRAM memory.

Parameters • **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- HAL status

### 50.2.15 HAL\_SDRAM\_Write\_8b

**Function Name** `HAL_StatusTypeDef HAL_SDRAM_Write_8b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)`

**Function Description** Writes 8-bit data buffer to SDRAM memory.

**Parameters**

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- HAL status

### 50.2.16 HAL\_SDRAM\_Read\_16b

**Function Name** `HAL_StatusTypeDef HAL_SDRAM_Read_16b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)`

**Function Description** Reads 16-bit data buffer from the SDRAM memory.

**Parameters**

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- HAL status

### 50.2.17 HAL\_SDRAM\_Write\_16b

**Function Name** `HAL_StatusTypeDef HAL_SDRAM_Write_16b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)`

**Function Description** Writes 16-bit data buffer to SDRAM memory.

**Parameters**

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- HAL status

### 50.2.18 HAL\_SDRAM\_Read\_32b

**Function Name** `HAL_StatusTypeDef HAL_SDRAM_Read_32b (SDRAM_HandleTypeDef * hsdram, uint32_t * pAddress,`

uint32\_t \* pDstBuffer, uint32\_t BufferSize)

Function Description Reads 32-bit data buffer from the SDRAM memory.

Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress**: Pointer to read start address
- **pDstBuffer**: Pointer to destination buffer
- **BufferSize**: Size of the buffer to read from memory

Return values

- HAL status

### 50.2.19 HAL\_SDRAM\_Write\_32b

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_Write\_32b**  
(SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress,  
uint32\_t \* pSrcBuffer, uint32\_t BufferSize)

Function Description Writes 32-bit data buffer to SDRAM memory.

Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress**: Pointer to write start address
- **pSrcBuffer**: Pointer to source buffer to write
- **BufferSize**: Size of the buffer to write to memory

Return values

- HAL status

### 50.2.20 HAL\_SDRAM\_Read\_DMA

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_Read\_DMA**  
(SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress,  
uint32\_t \* pDstBuffer, uint32\_t BufferSize)

Function Description Reads a Words data from the SDRAM memory using DMA transfer.

Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress**: Pointer to read start address
- **pDstBuffer**: Pointer to destination buffer
- **BufferSize**: Size of the buffer to read from memory

Return values

- HAL status

### 50.2.21 HAL\_SDRAM\_Write\_DMA

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_Write\_DMA**  
(SDRAM\_HandleTypeDef \* hsdram, uint32\_t \* pAddress,  
uint32\_t \* pSrcBuffer, uint32\_t BufferSize)

Function Description Writes a Words data buffer to SDRAM memory using DMA transfer.

Parameters

- **hsdram**: pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **pAddress**: Pointer to write start address
- **pSrcBuffer**: Pointer to source buffer to write
- **BufferSize**: Size of the buffer to write to memory



Return values

- HAL status

### 50.2.22 HAL\_SDRAM\_WriteProtection\_Enable

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_WriteProtection\_Enable (SDRAM\_HandleTypeDef \* hsdram)**

Function Description Enables dynamically SDRAM write protection.

Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- HAL status

### 50.2.23 HAL\_SDRAM\_WriteProtection\_Disable

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_WriteProtection\_Disable (SDRAM\_HandleTypeDef \* hsdram)**

Function Description Disables dynamically SDRAM write protection.

Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.

Return values

- HAL status

### 50.2.24 HAL\_SDRAM\_SendCommand

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_SendCommand (SDRAM\_HandleTypeDef \* hsdram, FMC\_SDRAM\_CommandTypeDef \* Command, uint32\_t Timeout)**

Function Description Sends Command to the SDRAM bank.

Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **Command:** SDRAM command structure
- **Timeout:** Timeout duration

Return values

- HAL status

### 50.2.25 HAL\_SDRAM\_ProgramRefreshRate

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_ProgramRefreshRate (SDRAM\_HandleTypeDef \* hsdram, uint32\_t RefreshRate)**

Function Description Programs the SDRAM Memory Refresh rate.

Parameters

- **hsdram:** pointer to a SDRAM\_HandleTypeDef structure that contains the configuration information for SDRAM module.
- **RefreshRate:** The SDRAM refresh rate value

Return values

- HAL status

### 50.2.26 HAL\_SDRAM\_SetAutoRefreshNumber

Function Name **HAL\_StatusTypeDef HAL\_SDRAM\_SetAutoRefreshNumber (SDRAM\_HandleTypeDef \* hsdram, uint32\_t AutoRefreshNumber)**

Function Description	Sets the Number of consecutive SDRAM Memory auto Refresh commands.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b>: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> <li>• <b>AutoRefreshNumber</b>: The SDRAM auto Refresh number</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 50.2.27 HAL\_SDRAM\_GetModeStatus

Function Name	<b>uint32_t HAL_SDRAM_GetModeStatus (SDRAM_HandleTypeDef * hsdram)</b>
Function Description	Returns the SDRAM memory current mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b>: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• The SDRAM memory mode.</li> </ul>

### 50.2.28 HAL\_SDRAM\_GetState

Function Name	<b>HAL_SDRAM_StateTypeDef HAL_SDRAM_GetState (SDRAM_HandleTypeDef * hsdram)</b>
Function Description	Returns the SDRAM state.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsdram</b>: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

## 50.3 SDRAM Firmware driver defines

### 50.3.1 SDRAM

#### *SDRAM Exported Macros*

<b>__HAL_SDRAM_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>• Reset SDRAM handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li>• <b>__HANDLE__</b>: specifies the SDRAM handle.</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>• None</li> </ul>
---------------------------------------	---

## 51 HAL SD Generic Driver

### 51.1 SD Firmware driver registers structures

#### 51.1.1 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*  
SDMMC 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

### 51.1.2 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 ManDeflECC`
- `__IO uint8_t WrSpeedFact`
- `__IO uint8_t MaxWrBlockLen`
- `__IO uint8_t WriteBlockPaPartial`
- `__IO uint8_t Reserved3`
- `__IO uint8_t ContentProtectAppli`
- `__IO uint8_t FileFormatGrouop`
- `__IO uint8_t CopyFlag`
- `__IO uint8_t PermWrProtect`
- `__IO uint8_t TempWrProtect`
- `__IO uint8_t FileFormat`
- `__IO uint8_t ECC`
- `__IO uint8_t CSD_CRC`
- `__IO uint8_t Reserved4`

#### Field Documentation

- `__IO uint8_t HAL_SD_CSDTypeDef::CSDStruct`  
CSD structure
- `__IO uint8_t HAL_SD_CSDTypeDef::SysSpecVersion`  
System specification version
- `__IO uint8_t HAL_SD_CSDTypeDef::Reserved1`  
Reserved

- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::TAAC**  
Data read access time 1
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::NSAC**  
Data read access time 2 in CLK cycles
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::MaxBusClkFrec**  
Max. bus clock frequency
- **\_\_IO uint16\_t HAL\_SD\_CSDTypeDef::CardComdClasses**  
Card command classes
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::RdBlockLen**  
Max. read data block length
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::PartBlockRead**  
Partial blocks for read allowed
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::WrBlockMisalign**  
Write block misalignment
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::RdBlockMisalign**  
Read block misalignment
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::DSRImpl**  
DSR implemented
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::Reserved2**  
Reserved
- **\_\_IO uint32\_t HAL\_SD\_CSDTypeDef::DeviceSize**  
Device Size
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::MaxRdCurrentVDDMin**  
Max. read current @ VDD min
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::MaxRdCurrentVDDMax**  
Max. read current @ VDD max
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::MaxWrCurrentVDDMin**  
Max. write current @ VDD min
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::MaxWrCurrentVDDMax**  
Max. write current @ VDD max
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::DeviceSizeMul**  
Device size multiplier
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::EraseGrSize**  
Erase group size
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::EraseGrMul**  
Erase group size multiplier
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::WrProtectGrSize**  
Write protect group size
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::WrProtectGrEnable**  
Write protect group enable
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::ManDeflECC**  
Manufacturer default ECC
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::WrSpeedFact**  
Write speed factor
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::MaxWrBlockLen**  
Max. write data block length
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::WriteBlockPaPartial**  
Partial blocks for write allowed
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::Reserved3**  
Reserved
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::ContentProtectAppli**  
Content protection application
- **\_\_IO uint8\_t HAL\_SD\_CSDTypeDef::FileFormatGroup**  
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

### 51.1.3 HAL\_SD\_CIDTypeDef

#### Data Fields

- **\_\_IO uint8\_t ManufacturerID**
- **\_\_IO uint16\_t OEM\_AppliID**
- **\_\_IO uint32\_t ProdName1**
- **\_\_IO uint8\_t ProdName2**
- **\_\_IO uint8\_t ProdRev**
- **\_\_IO uint32\_t ProdSN**
- **\_\_IO uint8\_t Reserved1**
- **\_\_IO uint16\_t ManufactDate**
- **\_\_IO uint8\_t CID\_CRC**
- **\_\_IO uint8\_t Reserved2**

#### Field Documentation

- **\_\_IO uint8\_t HAL\_SD\_CIDTypeDef::ManufacturerID**  
Manufacturer ID
- **\_\_IO uint16\_t HAL\_SD\_CIDTypeDef::OEM\_AppliID**  
OEM/Application ID
- **\_\_IO uint32\_t HAL\_SD\_CIDTypeDef::ProdName1**  
Product Name part1
- **\_\_IO uint8\_t HAL\_SD\_CIDTypeDef::ProdName2**  
Product Name part2
- **\_\_IO uint8\_t HAL\_SD\_CIDTypeDef::ProdRev**  
Product Revision
- **\_\_IO uint32\_t HAL\_SD\_CIDTypeDef::ProdSN**  
Product Serial Number
- **\_\_IO uint8\_t HAL\_SD\_CIDTypeDef::Reserved1**  
Reserved1
- **\_\_IO uint16\_t HAL\_SD\_CIDTypeDef::ManufactDate**  
Manufacturing Date
- **\_\_IO uint8\_t HAL\_SD\_CIDTypeDef::CID\_CRC**  
CID CRC

- `__IO uint8_t HAL_SD_CIDTypeDef::Reserved2`  
Always 1

#### 51.1.4 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

#### 51.1.5 HAL\_SD\_CardInfoTypeDef

##### Data Fields

- `HAL_SD_CSDTypeDef SD_csd`
- `HAL_SD_CIDTypeDef SD_cid`
- `uint64_t CardCapacity`
- `uint32_t CardBlockSize`

- ***uint16\_t RCA***
- ***uint8\_t CardType***

#### Field Documentation

- ***HAL\_SD\_CSDTypeDef HAL\_SD\_CardInfoTypeDef::SD\_csd***  
SD card specific data register
- ***HAL\_SD\_CIDTypeDef HAL\_SD\_CardInfoTypeDef::SD\_cid***  
SD card identification number register
- ***uint64\_t HAL\_SD\_CardInfoTypeDef::CardCapacity***  
Card capacity
- ***uint32\_t HAL\_SD\_CardInfoTypeDef::CardBlockSize***  
Card block size
- ***uint16\_t HAL\_SD\_CardInfoTypeDef::RCA***  
SD relative card address
- ***uint8\_t HAL\_SD\_CardInfoTypeDef::CardType***  
SD card type

## 51.2 SD Firmware driver API description

### 51.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in `HAL_SD_MspInit()` function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implement the `HAL_SD_MspInit()` API:
  - a. Enable the SDMMC interface clock using `__HAL_RCC_SDMMC_CLK_ENABLE();`
  - b. SDMMC pins configuration for SD card
    - Enable the clock for the SDMMC GPIOs using the functions `__HAL_RCC_GPIOx_CLK_ENABLE();`
    - Configure these SDMMC pins as alternate function pull-up using `HAL_GPIO_Init()` and according to your pin assignment;
  - c. 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 SDMMC and DMA interrupt priorities using functions `HAL_NVIC_SetPriority();` DMA priority is superior to SDMMC's priority
    - Enable the NVIC DMA and SDMMC IRQs using function `HAL_NVIC_EnableIRQ()`
    - SDMMC interrupts are managed using the macros `__HAL_SD_SDMMC_ENABLE_IT()` and `__HAL_SD_SDMMC_DISABLE_IT()` inside the communication process.



- SDMMC interrupts pending bits are managed using the macros `__HAL_SD_SDMMC_GET_IT()` and `__HAL_SD_SDMMC_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 (SDMMC\_CK) is computed as follows:  $SDMMC\_CK = SDMMCCLK / (ClockDiv + 2)$  In initialization mode and according to the SD Card standard, make sure that the SDMMC\_CK frequency doesn't exceed 400KHz.
2. Get the SD CID and CSD data. All these information are managed by the `SDCardInfo` structure. This structure provide also ready computed SD Card capacity and Block size. 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 SDMMC\_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 SDMMC 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

## 51.2.2 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\\_DeInit\(\)\*](#)
- [\*HAL\\_SD\\_MspInit\(\)\*](#)
- [\*HAL\\_SD\\_MspDeInit\(\)\*](#)

## 51.2.3 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\(\)\*](#)

## 51.2.4 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\(\)](#)

### 51.2.5 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\(\)](#)

### 51.2.6 HAL\_SD\_Init

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * SDCardInfo)</b>
Function Description	Initializes the SD card according to the specified parameters in the SD_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsd</b>: SD handle</li> <li>• <b>SDCardInfo</b>: HAL_SD_CardInfoTypeDef structure for SD card information</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL SD error state</li> </ul>

### 51.2.7 HAL\_SD\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)</b>
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>

### 51.2.8 HAL\_SD\_MspInit

Function Name	<b>void HAL_SD_MspInit (SD_HandleTypeDef * hsd)</b>
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>

### 51.2.9 HAL\_SD\_MspDeInit

Function Name	<b>void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)</b>
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>

**51.2.10 HAL\_SD\_ReadBlocks**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks</b> (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Reads block(s) from a specified address in a card.
Parameters	<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>

**51.2.11 HAL\_SD\_WriteBlocks**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks</b> (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Allows to write block(s) to a specified address in a card.
Parameters	<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>

**51.2.12 HAL\_SD\_ReadBlocks\_DMA**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks_DMA</b> (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Reads block(s) from a specified address in a card.
Parameters	<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>

**51.2.13 HAL\_SD\_WriteBlocks\_DMA**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks_DMA</b> (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Writes block(s) to a specified address in a card.
Parameters	<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>

**51.2.14 HAL\_SD\_CheckReadOperation**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_CheckReadOperation</b> (SD_HandleTypeDef * hsd, uint32_t Timeout)
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>

**51.2.15 HAL\_SD\_CheckWriteOperation**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_CheckWriteOperation</b> (SD_HandleTypeDef * hsd, uint32_t Timeout)
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>

**51.2.16 HAL\_SD\_Erase**

Function Name	<b>HAL_SD_ErrorTypeDef HAL_SD_Erase</b> (SD_HandleTypeDef * hsd, uint64_t startaddr, uint64_t endaddr)
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>

**51.2.17 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>

#### 51.2.18 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>

#### 51.2.19 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>

#### 51.2.20 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>

#### 51.2.21 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>

#### 51.2.22 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

### 51.2.23 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

### 51.2.24 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

### 51.2.25 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:  
SDMMC\_BUS\_WIDE\_8B: 8-bit data transfer (Only for MMC)  
SDMMC\_BUS\_WIDE\_4B: 4-bit data transfer  
SDMMC\_BUS\_WIDE\_1B: 1-bit data transfer

Return values

- SD Card error state

### 51.2.26 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

**51.2.27 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 SDMMCCK clock between 67 and 75 MHz</li> </ul>

**51.2.28 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>

**51.2.29 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>

**51.2.30 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>

**51.3 SD Firmware driver defines****51.3.1 SD*****SD Exported Constants***

<b>SD_CMD_GO_IDLE_STATE</b>	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_SDMMC_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. Sets the address of the first write block to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE	Reserved for SD security applications.
SD_CMD_FAST_IO	SD card doesn't support it (Reserved).
SD_CMD_GO_IRQ_STATE	SD card doesn't support it (Reserved).
SD_CMD_LOCK_UNLOCK	Sets/resets the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
SD_CMD_APP_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
SD_CMD_GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general purpose/application specific commands.
SD_CMD_NO_CMD	
SD_CMD_APP_SD_SET_BUSWIDTH	SDMMC_APP_CMD should be sent before sending these commands. (ACMD6) Defines the data bus width to be used for data transfer. The allowed data bus widths are given in SCR register.
SD_CMD_SD_APP_STATUS	(ACMD13) Sends the SD status.
SD_CMD_SD_APP_SEND_NUM_WRITE_BLOCKS	(ACMD22) Sends the number of the written (without errors) write blocks.

	Responds with 32bit+CRC data block.
SD_CMD_SD_APP_OP_COND	(ACMD41) Sends host capacity support information (HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_SD_APP_SET_CLR_CARD_DETECT	(ACMD42) Connects/Disconnects the 50 KOhm pull-up resistor on CD/DAT3 (pin 1) of the card.
SD_CMD_SD_APP_SEND_SCR	Reads the SD Configuration Register (SCR).
SD_CMD_SDMMC_RW_DIRECT	For SD I/O card only, reserved for security specification.
SD_CMD_SDMMC_RW_EXTENDED	For SD I/O card only, reserved for security specification.
SD_CMD_SD_APP_GET_MKB	SD_CMD_APP_CMD should be sent before sending these commands. For SD card only
SD_CMD_SD_APP_GET_MID	For SD card only
SD_CMD_SD_APP_SET_CER_RN1	For SD card only
SD_CMD_SD_APP_GET_CER_RN2	For SD card only
SD_CMD_SD_APP_SET_CER_RES2	For SD card only
SD_CMD_SD_APP_GET_CER_RES1	For SD card only
SD_CMD_SD_APP_SECURE_READ_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_ERASE	For SD card only
SD_CMD_SD_APP_CHANGE_SECURE_AREA	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MKB	For SD card only
STD_CAPACITY_SD_CARD_V1_1	
STD_CAPACITY_SD_CARD_V2_0	
HIGH_CAPACITY_SD_CARD	
MULTIMEDIA_CARD	
SECURE_DIGITAL_IO_CARD	
HIGH_SPEED_MULTIMEDIA_CARD	
SECURE_DIGITAL_IO_COMBO_CARD	
HIGH_CAPACITY_MMC_CARD	

**SD Exported Macros****\_\_HAL\_SD\_SDMMC\_ENABLE****Description:**

- Enable the SD device.

**Return value:**

- None

**\_\_HAL\_SD\_SDMMC\_DISABLE****Description:**

- Disable the SD device.

**Return value:**

- None

**\_\_HAL\_SD\_SDMMC\_DMA\_ENABLE****Description:**

- Enable the SDMMC DMA transfer.

**Return value:**

- None

**\_\_HAL\_SD\_SDMMC\_DMA\_DISABLE****Description:**

- Disable the SDMMC DMA transfer.

**Return value:**

- None

**\_\_HAL\_SD\_SDMMC\_ENABLE\_IT****Description:**

- Enable the SD device interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: SD Handle
- **\_\_INTERRUPT\_\_**: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - **SDMMC\_IT\_CCRCFAIL**: Command response received (CRC check failed) interrupt
  - **SDMMC\_IT\_DCRCFAIL**: Data block sent/received (CRC check failed) interrupt
  - **SDMMC\_IT\_CTIMEOUT**: Command response timeout interrupt
  - **SDMMC\_IT\_DTIMEOUT**: Data timeout interrupt
  - **SDMMC\_IT\_TXUNDERR**: Transmit FIFO underrun error interrupt
  - **SDMMC\_IT\_RXOVERR**: Received FIFO overrun error interrupt
  - **SDMMC\_IT\_CMDREND**: Command response received (CRC check passed) interrupt
  - **SDMMC\_IT\_CMDSSENT**: Command sent (no response required) interrupt

- SDMMC\_IT\_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDMMC\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDMMC\_IT\_CMDACT: Command transfer in progress interrupt
- SDMMC\_IT\_TXACT: Data transmit in progress interrupt
- SDMMC\_IT\_RXACT: Data receive in progress interrupt
- SDMMC\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDMMC\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
- SDMMC\_IT\_TXFIFO: Transmit FIFO full interrupt
- SDMMC\_IT\_RXFIFO: Receive FIFO full interrupt
- SDMMC\_IT\_TXFIFOE: Transmit FIFO empty interrupt
- SDMMC\_IT\_RXFIFOE: Receive FIFO empty interrupt
- SDMMC\_IT\_TXDAVL: Data available in transmit FIFO interrupt
- SDMMC\_IT\_RXDAVL: Data available in receive FIFO interrupt
- SDMMC\_IT\_SDIOIT: SD I/O interrupt received interrupt

**Return value:**

- None

**\_\_HAL\_SD\_SDMMC\_DISABLE\_IT****Description:**

- Disable the SD device interrupt.

**Parameters:**

- \_\_HANDLE\_\_: SD Handle
- \_\_INTERRUPT\_\_: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDMMC\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC\_IT\_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC\_IT\_CTIMEOUT: Command response timeout interrupt
  - SDMMC\_IT\_DTIMEOUT: Data timeout interrupt
  - SDMMC\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt

- SDMMC\_IT\_RXOVERR: Received FIFO overrun error interrupt
- SDMMC\_IT\_CMDREND: Command response received (CRC check passed) interrupt
- SDMMC\_IT\_CMDSSENT: Command sent (no response required) interrupt
- SDMMC\_IT\_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDMMC\_IT\_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDMMC\_IT\_CMDACT: Command transfer in progress interrupt
- SDMMC\_IT\_TXACT: Data transmit in progress interrupt
- SDMMC\_IT\_RXACT: Data receive in progress interrupt
- SDMMC\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDMMC\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
- SDMMC\_IT\_TXFIFO: Transmit FIFO full interrupt
- SDMMC\_IT\_RXFIFO: Receive FIFO full interrupt
- SDMMC\_IT\_TXFIFOE: Transmit FIFO empty interrupt
- SDMMC\_IT\_RXFIFOE: Receive FIFO empty interrupt
- SDMMC\_IT\_TXDAVL: Data available in transmit FIFO interrupt
- SDMMC\_IT\_RXDAVL: Data available in receive FIFO interrupt
- SDMMC\_IT\_SDIOIT: SD I/O interrupt received interrupt

**Return value:**

- None

**\_\_HAL\_SD\_SDMMC\_GET\_FLAG****Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: SD Handle
- \_\_FLAG\_\_: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC\_FLAG\_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC\_FLAG\_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC\_FLAG\_CTIMEOUT: Command response timeout

- SDMMC\_FLAG\_DTIMEOUT: Data timeout
- SDMMC\_FLAG\_TXUNDERR: Transmit FIFO underrun error
- SDMMC\_FLAG\_RXOVERR: Received FIFO overrun error
- SDMMC\_FLAG\_CMDREND: Command response received (CRC check passed)
- SDMMC\_FLAG\_CMDSSENT: Command sent (no response required)
- SDMMC\_FLAG\_DATAEND: Data end (data counter, SDIDCOUNT, is zero)
- SDMMC\_FLAG\_DBCKEND: Data block sent/received (CRC check passed)
- SDMMC\_FLAG\_CMDACT: Command transfer in progress
- SDMMC\_FLAG\_TXACT: Data transmit in progress
- SDMMC\_FLAG\_RXACT: Data receive in progress
- SDMMC\_FLAG\_TXFIFOHE: Transmit FIFO Half Empty
- SDMMC\_FLAG\_RXFIFOHF: Receive FIFO Half Full
- SDMMC\_FLAG\_TXFIFO: Transmit FIFO full
- SDMMC\_FLAG\_RXFIFO: Receive FIFO full
- SDMMC\_FLAG\_TXFIFOE: Transmit FIFO empty
- SDMMC\_FLAG\_RXFIFOE: Receive FIFO empty
- SDMMC\_FLAG\_TXDAVL: Data available in transmit FIFO
- SDMMC\_FLAG\_RXDAVL: Data available in receive FIFO
- SDMMC\_FLAG\_SDIOIT: SD I/O interrupt received

**Return value:**

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

**\_\_HAL\_SD\_SDMMC\_CLEAR\_FLAG****Description:**

- Clear the SD's pending flags.

**Parameters:**

- \_\_HANDLE\_\_: SD Handle
- \_\_FLAG\_\_: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDMMC\_FLAG\_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC\_FLAG\_DCRCFAIL: Data block sent/received (CRC check failed)



- SDMMC\_FLAG\_CTIMEOUT: Command response timeout
- SDMMC\_FLAG\_DTIMEOUT: Data timeout
- SDMMC\_FLAG\_TXUNDERR: Transmit FIFO underrun error
- SDMMC\_FLAG\_RXOVERR: Received FIFO overrun error
- SDMMC\_FLAG\_CMDREND: Command response received (CRC check passed)
- SDMMC\_FLAG\_CMDSSENT: Command sent (no response required)
- SDMMC\_FLAG\_DATAEND: Data end (data counter, SDIDCOUNT, is zero)
- SDMMC\_FLAG\_DBCKEND: Data block sent/received (CRC check passed)
- SDMMC\_FLAG\_SDIOIT: SD I/O interrupt received

**Return value:**

- None

**\_\_HAL\_SD\_SDMMC\_GET\_IT****Description:**

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

**Parameters:**

- \_\_HANDLE\_\_: SD Handle
- \_\_INTERRUPT\_\_: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC\_IT\_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC\_IT\_CTIMEOUT: Command response timeout interrupt
  - SDMMC\_IT\_DTIMEOUT: Data timeout interrupt
  - SDMMC\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC\_IT\_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC\_IT\_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC\_IT\_CMDSSENT: Command sent (no response required) interrupt
  - SDMMC\_IT\_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
  - SDMMC\_IT\_DBCKEND: Data block sent/received (CRC check passed)

- interrupt
- SDMMC\_IT\_CMDACT: Command transfer in progress interrupt
- SDMMC\_IT\_TXACT: Data transmit in progress interrupt
- SDMMC\_IT\_RXACT: Data receive in progress interrupt
- SDMMC\_IT\_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDMMC\_IT\_RXFIFOHF: Receive FIFO Half Full interrupt
- SDMMC\_IT\_TXFIFO: Transmit FIFO full interrupt
- SDMMC\_IT\_RXFIFO: Receive FIFO full interrupt
- SDMMC\_IT\_TXFIFOE: Transmit FIFO empty interrupt
- SDMMC\_IT\_RXFIFOE: Receive FIFO empty interrupt
- SDMMC\_IT\_TXDAVL: Data available in transmit FIFO interrupt
- SDMMC\_IT\_RXDAVL: Data available in receive FIFO interrupt
- SDMMC\_IT\_SDIOIT: SD I/O interrupt received interrupt

**Return value:**

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

**\_\_HAL\_SD\_SDMMC\_CLEAR\_IT****Description:**

- Clear the SD's interrupt pending bits.

**Parameters:**

- \_\_HANDLE\_\_: SD Handle
- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDMMC\_IT\_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC\_IT\_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC\_IT\_CTIMEOUT: Command response timeout interrupt
  - SDMMC\_IT\_DTIMEOUT: Data timeout interrupt
  - SDMMC\_IT\_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC\_IT\_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC\_IT\_CMDREND: Command response received (CRC check passed) interrupt

- SDMMC\_IT\_CMDSENT: Command sent (no response required) interrupt
- SDMMC\_IT\_DATAEND: Data end (data counter, SDMMC\_DCOUNT, is zero) interrupt
- SDMMC\_IT\_SDIOIT: SD I/O interrupt received interrupt

**Return value:**

- None

***SD Handle Structure definition***

SD\_InitTypeDef

SD\_TypeDef

***SD Private Defines***

DATA\_BLOCK\_SIZE

SDMMC\_STATIC\_FLAGS

SDMMC\_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\_SDMMC\_SEND\_IF\_COND

SDMMC\_APP\_CMD should be sent before sending these commands.

## 52 HAL SMARTCARD Generic Driver

### 52.1 SMARTCARD Firmware driver registers structures

#### 52.1.1 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 OneBitSampling*
- *uint32\_t Prescaler*
- *uint32\_t GuardTime*
- *uint32\_t NACKState*
- *uint32\_t TimeOutEnable*
- *uint32\_t TimeOutValue*
- *uint32\_t BlockLength*
- *uint32\_t AutoRetryCount*

##### Field Documentation

- ***uint32\_t SMARTCARD\_InitTypeDef::BaudRate***  
Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((PCLKx) / ((hsc->Init.BaudRate)))
- ***uint32\_t SMARTCARD\_InitTypeDef::WordLength***  
Specifies the number of data bits transmitted or received in a frame. This parameter [SMARTCARD\\_Word\\_Length](#) can only be set to 9 (8 data + 1 parity bits).
- ***uint32\_t SMARTCARD\_InitTypeDef::StopBits***  
Specifies the number of stop bits [SMARTCARD\\_Stop\\_Bits](#). Only 1.5 stop bits are authorized in SmartCard mode.
- ***uint32\_t SMARTCARD\_InitTypeDef::Parity***  
Specifies the parity mode. This parameter can be a value of [SMARTCARD\\_Parity](#)  
**Note:** The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- ***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::OneBitSampling***  
Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of [SMARTCARD\\_OneBit\\_Sampling](#)
- ***uint32\_t SMARTCARD\_InitTypeDef::Prescaler***  
Specifies the SmartCard Prescaler
- ***uint32\_t SMARTCARD\_InitTypeDef::GuardTime***  
Specifies the SmartCard Guard Time
- ***uint32\_t SMARTCARD\_InitTypeDef::NACKState***  
Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of [SMARTCARD\\_NACK\\_State](#)
- ***uint32\_t SMARTCARD\_InitTypeDef::TimeoutEnable***  
Specifies whether the receiver timeout is enabled. This parameter can be a value of [SMARTCARD\\_Timeout\\_Enable](#)
- ***uint32\_t SMARTCARD\_InitTypeDef::TimeoutValue***  
Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint32\_t SMARTCARD\_InitTypeDef::BlockLength***  
Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint32\_t SMARTCARD\_InitTypeDef::AutoRetryCount***  
Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)

### 52.1.2 SMARTCARD\_AdvFeatureInitTypeDef

#### Data Fields

- ***uint32\_t AdvFeatureInit***
- ***uint32\_t TxPinLevelInvert***
- ***uint32\_t RxPinLevelInvert***
- ***uint32\_t DataInvert***
- ***uint32\_t Swap***
- ***uint32\_t OverrunDisable***
- ***uint32\_t DMADisableonRxError***
- ***uint32\_t MSBFirst***

#### Field Documentation

- ***uint32\_t SMARTCARD\_AdvFeatureInitTypeDef::AdvFeatureInit***  
Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of [SMARTCARD\\_Advanced\\_Features\\_Initialization\\_Type](#)
- ***uint32\_t SMARTCARD\_AdvFeatureInitTypeDef::TxPinLevelInvert***  
Specifies whether the TX pin active level is inverted. This parameter can be a value of [SMARTCARD\\_Tx\\_Inv](#)

- ***uint32\_t SMARTCARD\_AdvFeatureInitTypeDef::RxPinLevelInvert***  
Specifies whether the RX pin active level is inverted. This parameter can be a value of [SMARTCARD\\_Rx\\_Inv](#)
- ***uint32\_t SMARTCARD\_AdvFeatureInitTypeDef::DataInvert***  
Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of [SMARTCARD\\_Data\\_Inv](#)
- ***uint32\_t SMARTCARD\_AdvFeatureInitTypeDef::Swap***  
Specifies whether TX and RX pins are swapped. This parameter can be a value of [SMARTCARD\\_Rx\\_Tx\\_Swap](#)
- ***uint32\_t SMARTCARD\_AdvFeatureInitTypeDef::OverrunDisable***  
Specifies whether the reception overrun detection is disabled. This parameter can be a value of [SMARTCARD\\_Overrun\\_Disable](#)
- ***uint32\_t SMARTCARD\_AdvFeatureInitTypeDef::DMADisableonRxError***  
Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of [SMARTCARD\\_DMA\\_Disable\\_on\\_Rx\\_Error](#)
- ***uint32\_t SMARTCARD\_AdvFeatureInitTypeDef::MSBFirst***  
Specifies whether MSB is sent first on UART line. This parameter can be a value of [SMARTCARD\\_MSB\\_First](#)

### 52.1.3 SMARTCARD\_HandleTypeDef

#### Data Fields

- ***USART\_TypeDef \* Instance***
- ***SMARTCARD\_InitTypeDef Init***
- ***SMARTCARD\_AdvFeatureInitTypeDef AdvancedInit***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferSize***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***\_\_IO HAL\_SMARTCARD\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***USART\_TypeDef\* SMARTCARD\_HandleTypeDef::Instance***
- ***SMARTCARD\_InitTypeDef SMARTCARD\_HandleTypeDef::Init***
- ***SMARTCARD\_AdvFeatureInitTypeDef SMARTCARD\_HandleTypeDef::AdvancedInit***
- ***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`

## 52.2 SMARTCARD Firmware driver API description

### 52.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follow:

1. Declare a SMARTCARD\_HandleTypeDef handle structure.
2. Associate a USART to the SMARTCARD handle hsc.
3. 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.
    - The specific USART 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.
  - 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.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsc Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsc AdvancedInit structure.
6. Initialize the SMARTCARD associated USART registers by calling the HAL\_SMARTCARD\_Init() API.



HAL\_SMARTCARD\_Init() API also configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL\_SMARTCARD\_MspInit() API.



## 52.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART associated to the SmartCard.

- These parameters can be configured:
  - Baud Rate
  - Parity: parity should be enabled, Frame Length is fixed to 8 bits plus parity: the USART frame format is given in [Table 16: "USART frame formats"](#).
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

**Table 16: USART frame formats**

M1 M0 bits	PCE bit	USART frame
01	1	SB   8 bit data   PB   STB

The HAL\_SMARTCARD\_Init() API follow respectively the USART (a)synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

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

## 52.2.3 IO operation functions

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\(\)](#)

## 52.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL\_SMARTCARD\_GetState() API is helpful to check in run-time the state of the SMARTCARD peripheral
- SMARTCARD\_SetConfig() API configures the SMARTCARD peripheral
- SMARTCARD\_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization

This section contains the following APIs:

- [HAL\\_SMARTCARD\\_GetState\(\)](#)
- [HAL\\_SMARTCARD\\_GetError\(\)](#)

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

## 52.2.6 HAL\_SMARTCARD\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	DeInitializes the SMARTCARD peripheral.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b>: SMARTCARD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

## 52.2.7 HAL\_SMARTCARD\_MspInit

Function Name	<b>void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD MSP Init.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b>: SMARTCARD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 52.2.8 HAL\_SMARTCARD\_MspDeInit

Function Name	<b>void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD MSP DeInit.
Parameters	<ul style="list-style-type: none"><li>• <b>hsc</b>: SMARTCARD handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

## 52.2.9 HAL\_SMARTCARD\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit</b> (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: SMARTCARD handle</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>

### 52.2.10 HAL\_SMARTCARD\_Receive

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive</b> (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: SMARTCARD handle</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>

### 52.2.11 HAL\_SMARTCARD\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT</b> (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: SMARTCARD handle</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>

### 52.2.12 HAL\_SMARTCARD\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT</b> (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: SMARTCARD handle</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>

### 52.2.13 HAL\_SMARTCARD\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA</b> (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t
---------------	--

	<b>Size)</b>
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: SMARTCARD handle</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>
<b>52.2.14 HAL_SMARTCARD_Receive_DMA</b>	
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 DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: SMARTCARD handle</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>• The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li> </ul>
<b>52.2.15 HAL_SMARTCARD_IRQHandler</b>	
Function Name	<b>void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsc)</b>
Function Description	SMARTCARD interrupt requests handling.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc</b>: SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
<b>52.2.16 HAL_SMARTCARD_TxCpltCallback</b>	
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>: SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
<b>52.2.17 HAL_SMARTCARD_RxCpltCallback</b>	
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>: SMARTCARD handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>
<b>52.2.18 HAL_SMARTCARD_ErrorCallback</b>	

Function Name      **void HAL\_SMARTCARD\_ErrorCallback  
(SMARTCARD\_HandleTypeDef \* hsc)**

Function Description    SMARTCARD error callbacks.

Parameters              •    **hsc:** SMARTCARD handle

Return values            •    None

### 52.2.19 HAL\_SMARTCARD\_GetState

Function Name      **HAL\_SMARTCARD\_StateTypeDef  
HAL\_SMARTCARD\_GetState (SMARTCARD\_HandleTypeDef \*  
hsc)**

Function Description    return the SMARTCARD state

Parameters              •    **hsc:** SMARTCARD handle

Return values            •    HAL state

### 52.2.20 HAL\_SMARTCARD\_GetError

Function Name      **uint32\_t HAL\_SMARTCARD\_GetError  
(SMARTCARD\_HandleTypeDef \* hsc)**

Function Description    Return the SMARTCARD error code.

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

Return values            •    SMARTCARD Error Code

## 52.3 SMARTCARD Firmware driver defines

### 52.3.1 SMARTCARD

#### ***SMARTCARD Advanced Features Initialization Type***

SMARTCARD\_ADVFEATURE\_NO\_INIT

SMARTCARD\_ADVFEATURE\_TXINVERT\_INIT

SMARTCARD\_ADVFEATURE\_RXINVERT\_INIT

SMARTCARD\_ADVFEATURE\_DATAINVERT\_INIT

SMARTCARD\_ADVFEATURE\_SWAP\_INIT

SMARTCARD\_ADVFEATURE\_RXOVERRUNDISABLE\_INIT

SMARTCARD\_ADVFEATURE\_DMADISABLEONERROR\_INIT

SMARTCARD\_ADVFEATURE\_MSBFIRST\_INIT

#### ***SMARTCARD Clock Phase***

SMARTCARD\_PHASE\_1EDGE

SMARTCARD\_PHASE\_2EDGE

#### ***SMARTCARD Clock Polarity***

SMARTCARD\_POLARITY\_LOW

SMARTCARD\_POLARITY\_HIGH

**SMARTCARD CR3 SCAR CNT LSB POS**

SMARTCARD\_CR3\_SCARCNT\_LSB\_POS

**SMARTCARD Data Inv**

SMARTCARD\_ADVFEATURE\_DATAINV\_DISABLE

SMARTCARD\_ADVFEATURE\_DATAINV\_ENABLE

**SMARTCARD DMA Disable on Rx Error**

SMARTCARD\_ADVFEATURE\_DMA\_ENABLEONRXERROR

SMARTCARD\_ADVFEATURE\_DMA\_DISABLEONRXERROR

**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
HAL_SMARTCARD_ERROR_RTO	Receiver TimeOut error

**SMARTCARD Exported Macros**

\_\_HAL\_SMARTCARD\_RESET\_HANDLE\_STATE

**Description:**

- Reset SMARTCARD handle state.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2

**Return value:**

- None

\_\_HAL\_SMARTCARD\_FLUSH\_DRREGISTER

**Description:**

- Flush the Smartcard DR register.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

**\_\_HAL\_SMARTCARD\_GET\_FLAG**

- None

**Description:**

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - SMARTCARD\_FLAG\_REACK: Receive enable acknowledge flag
  - SMARTCARD\_FLAG\_TEACK: Transmit enable acknowledge flag
  - SMARTCARD\_FLAG\_BUSY: Busy flag
  - SMARTCARD\_FLAG\_EOBF: End of block flag
  - SMARTCARD\_FLAG\_RTOF: Receiver timeout flag
  - SMARTCARD\_FLAG\_TXE: Transmit data register empty flag
  - SMARTCARD\_FLAG\_TC: Transmission Complete flag
  - SMARTCARD\_FLAG\_RXNE: Receive data register not empty flag
  - SMARTCARD\_FLAG\_ORE: OverRun Error flag
  - SMARTCARD\_FLAG\_NE: Noise Error flag
  - SMARTCARD\_FLAG\_FE: Framing Error flag
  - SMARTCARD\_FLAG\_PE: Parity Error flag

**Return value:**

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

**\_\_HAL\_SMARTCARD\_ENABLE\_IT****Description:**

- Enables the specified SmartCard interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- **\_\_INTERRUPT\_\_**: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_EOBF: End Of

- Block interrupt
- SMARTCARD\_IT\_RTOF: Receive TimeOut interrupt
- SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
- SMARTCARD\_IT\_TC: Transmission complete interrupt
- SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
- SMARTCARD\_IT\_PE: Parity Error interrupt
- SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

**Description:**

- Disables the specified SmartCard interrupt.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `__INTERRUPT__`: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD\_IT\_EOBF: End Of Block interrupt
  - SMARTCARD\_IT\_RTOF: Receive TimeOut interrupt
  - SMARTCARD\_IT\_TXE: Transmit Data Register empty interrupt
  - SMARTCARD\_IT\_TC: Transmission complete interrupt
  - SMARTCARD\_IT\_RXNE: Receive Data register not empty interrupt
  - SMARTCARD\_IT\_PE: Parity Error interrupt
  - SMARTCARD\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

**Description:**

- Checks whether the specified SmartCard interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD

`__HAL_SMARTCARD_DISABLE_IT`

`__HAL_SMARTCARD_GET_IT`



Handle. The Handle Instance which can be USART1 or USART2.

- `__IT__`: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - `SMARTCARD_IT_EOBF`: End Of Block interrupt
  - `SMARTCARD_IT_RTOF`: Receive TimeOut interrupt
  - `SMARTCARD_IT_TXE`: Transmit Data Register empty interrupt
  - `SMARTCARD_IT_TC`: Transmission complete interrupt
  - `SMARTCARD_IT_RXNE`: Receive Data register not empty interrupt
  - `SMARTCARD_IT_ORE`: OverRun Error interrupt
  - `SMARTCARD_IT_NE`: Noise Error interrupt
  - `SMARTCARD_IT_FE`: Framing Error interrupt
  - `SMARTCARD_IT_PE`: Parity Error interrupt

**Return value:**

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

`__HAL_SMARTCARD_GET_IT_SOURCE`

**Description:**

- Checks whether the specified SmartCard interrupt interrupt source is enabled.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `__IT__`: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - `SMARTCARD_IT_EOBF`: End Of Block interrupt
  - `SMARTCARD_IT_RTOF`: Receive TimeOut interrupt
  - `SMARTCARD_IT_TXE`: Transmit Data Register empty interrupt
  - `SMARTCARD_IT_TC`: Transmission complete interrupt
  - `SMARTCARD_IT_RXNE`: Receive Data register not empty interrupt
  - `SMARTCARD_IT_ORE`: OverRun Error interrupt
  - `SMARTCARD_IT_NE`: Noise Error interrupt
  - `SMARTCARD_IT_FE`: Framing Error

- interrupt
- SMARTCARD\_IT\_PE: Parity Error interrupt

**Return value:**

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

**\_\_HAL\_SMARTCARD\_CLEAR\_IT****Description:**

- Clears the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- \_\_IT\_CLEAR\_\_: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - USART\_CLEAR\_PEF: Parity Error Clear Flag
  - USART\_CLEAR\_FEF: Framing Error Clear Flag
  - USART\_CLEAR\_NEF: Noise detected Clear Flag
  - USART\_CLEAR\_OREF: OverRun Error Clear Flag
  - USART\_CLEAR\_TCF: Transmission Complete Clear Flag
  - USART\_CLEAR\_RTOF: Receiver Time Out Clear Flag
  - USART\_CLEAR\_EOBF: End Of Block Clear Flag

**Return value:**

- None

**\_\_HAL\_SMARTCARD\_SEND\_REQ****Description:**

- Set a specific SMARTCARD request flag.

**Parameters:**

- \_\_HANDLE\_\_: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- \_\_REQ\_\_: specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD\_RXDATA\_FLUSH\_REQUEST: Receive Data flush Request
  - SMARTCARD\_TXDATA\_FLUSH\_REQUEST: Transmit data flush Request

`__HAL_SMARTCARD_ENABLE`

**Return value:**

- None

**Description:**

- Enable the USART associated to the SMARTCARD Handle.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

**Description:**

- Disable the USART associated to the SMARTCARD Handle.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.

**Return value:**

- None

**Description:**

- Macros to enable or disable the SmartCard DMA request.

**Parameters:**

- `__HANDLE__`: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- `__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_ENABLE`

`__HAL_SMARTCARD_DMA_REQUEST_DISABLE`

**SMARTCARD GTPR GT LSBPOS**

`SMARTCARD_GTPR_GT_LSB_POS`

**SMARTCARD Interruption Mask**

`SMARTCARD_IT_MASK`

**SMARTCARD Interruption definition**

`SMARTCARD_IT_PE`

SMARTCARD\_IT\_TXE  
SMARTCARD\_IT\_TC  
SMARTCARD\_IT\_RXNE  
SMARTCARD\_IT\_ERR  
SMARTCARD\_IT\_ORE  
SMARTCARD\_IT\_NE  
SMARTCARD\_IT\_FE  
SMARTCARD\_IT\_EOB  
SMARTCARD\_IT\_RTO

**SMARTCARD IT CLEAR Flags**

SMARTCARD\_CLEAR\_PEF     Parity Error Clear Flag  
SMARTCARD\_CLEAR\_FEF     Framing Error Clear Flag  
SMARTCARD\_CLEAR\_NEF     Noise detected Clear Flag  
SMARTCARD\_CLEAR\_OREF    OverRun Error Clear Flag  
SMARTCARD\_CLEAR\_TCF     Transmission Complete Clear Flag  
SMARTCARD\_CLEAR\_RTOF    Receiver Time Out Clear Flag  
SMARTCARD\_CLEAR\_EOBF    End Of Block Clear Flag

**SMARTCARD Last Bit**

SMARTCARD\_LASTBIT\_DISABLE  
SMARTCARD\_LASTBIT\_ENABLE

**SMARTCARD Mode**

SMARTCARD\_MODE\_RX  
SMARTCARD\_MODE\_TX  
SMARTCARD\_MODE\_TX\_RX

**SMARTCARD MSB First**

SMARTCARD\_ADVFEATURE\_MSBFIRST\_DISABLE  
SMARTCARD\_ADVFEATURE\_MSBFIRST\_ENABLE

**SMARTCARD NACK State**

SMARTCARD\_NACK\_ENABLE  
SMARTCARD\_NACK\_DISABLE

**SMARTCARD OneBit Sampling**

SMARTCARD\_ONE\_BIT\_SAMPLE\_DISABLE  
SMARTCARD\_ONE\_BIT\_SAMPLE\_ENABLE

**SMARTCARD Overrun Disable**

SMARTCARD\_ADVFEATURE\_OVERRUN\_ENABLE  
SMARTCARD\_ADVFEATURE\_OVERRUN\_DISABLE

**SMARTCARD Parity**

SMARTCARD\_PARITY\_EVEN

SMARTCARD\_PARITY\_ODD

**SMARTCARD Private Constants**

TEACK\_REACK\_TIMEOUT

HAL\_SMARTCARD\_TXDMA\_TIMEOUTVALUE

USART\_CR1\_FIELDS

USART\_CR2\_CLK\_FIELDS

USART\_CR2\_FIELDS

USART\_CR3\_FIELDS

IS\_SMARTCARD\_WORD\_LENGTH

IS\_SMARTCARD\_STOPBITS

IS\_SMARTCARD\_PARITY

IS\_SMARTCARD\_MODE

IS\_SMARTCARD\_POLARITY

IS\_SMARTCARD\_PHASE

IS\_SMARTCARD\_LASTBIT

IS\_SMARTCARD\_ONE\_BIT\_SAMPLE

IS\_SMARTCARD\_NACK

IS\_SMARTCARD\_TIMEOUT

IS\_SMARTCARD\_ADVFEATURE\_INIT

IS\_SMARTCARD\_ADVFEATURE\_TXINV

IS\_SMARTCARD\_ADVFEATURE\_RXINV

IS\_SMARTCARD\_ADVFEATURE\_DATAINV

IS\_SMARTCARD\_ADVFEATURE\_SWAP

IS\_SMARTCARD\_OVERRUN

IS\_SMARTCARD\_ADVFEATURE\_DMAONRXERROR

IS\_SMARTCARD\_BAUDRATE

IS\_SMARTCARD\_BLOCKLENGTH

IS\_SMARTCARD\_TIMEOUT\_VALUE

IS\_SMARTCARD\_AUTORETRY\_COUNT

IS\_SMARTCARD\_ADVFEATURE\_MSBFIRST

IS\_SMARTCARD\_REQUEST\_PARAMETER

**SMARTCARD Request Parameters**

SMARTCARD\_RXDATA\_FLUSH\_REQUEST    Receive Data flush Request

SMARTCARD\_TXDATA\_FLUSH\_REQUEST    Transmit data flush Request

**SMARTCARD RTOR BLEN LSBPOS**

SMARTCARD\_RTOR\_BLEN\_LSB\_POS

**SMARTCARD Rx Inv**

SMARTCARD\_ADVFEATURE\_RXINV\_DISABLE

SMARTCARD\_ADVFEATURE\_RXINV\_ENABLE

**SMARTCARD Rx Tx Swap**

SMARTCARD\_ADVFEATURE\_SWAP\_DISABLE

SMARTCARD\_ADVFEATURE\_SWAP\_ENABLE

**SMARTCARD Number of Stop Bits**

SMARTCARD\_STOPBITS\_1\_5

**SMARTCARD Timeout Enable**

SMARTCARD\_TIMEOUT\_DISABLE

SMARTCARD\_TIMEOUT\_ENABLE

**SMARTCARD Tx Inv**

SMARTCARD\_ADVFEATURE\_TXINV\_DISABLE

SMARTCARD\_ADVFEATURE\_TXINV\_ENABLE

**SMARTCARD Word Length**

SMARTCARD\_WORDLENGTH\_9B

## 53 HAL SMARTCARD Extension Driver

### 53.1 SMARTCARDEx Firmware driver API description

#### 53.1.1 How to use this driver

The Extended SMARTCARD HAL driver can be used as follow:

1. After having configured the SMARTCARD basic features with `HAL_SMARTCARD_Init()`, then if required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the `hsc AdvancedInit` structure.

#### 53.1.2 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- `HAL_SMARTCARDEx_BlockLength_Config()` API allows to configure the Block Length on the fly
- `HAL_SMARTCARDEx_TimeOut_Config()` API allows to configure the receiver timeout value on the fly
- `HAL_SMARTCARDEx_EnableReceiverTimeOut()` API enables the receiver timeout feature
- `HAL_SMARTCARDEx_DisableReceiverTimeOut()` API disables the receiver timeout feature

This section contains the following APIs:

- [`HAL\_SMARTCARDEx\_BlockLength\_Config\(\)`](#)
- [`HAL\_SMARTCARDEx\_TimeOut\_Config\(\)`](#)
- [`HAL\_SMARTCARDEx\_EnableReceiverTimeOut\(\)`](#)
- [`HAL\_SMARTCARDEx\_DisableReceiverTimeOut\(\)`](#)

#### 53.1.3 HAL\_SMARTCARDEx\_BlockLength\_Config

Function Name	<b><code>void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsc, uint8_t BlockLength)</code></b>
Function Description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>BlockLength:</b> SMARTCARD block length (8-bit long at most)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

#### 53.1.4 HAL\_SMARTCARDEx\_TimeOut\_Config

Function Name	<b><code>void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsc, uint32_t TimeOutValue)</code></b>
Function Description	Update on the fly the receiver timeout value in RTOR register.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsc:</b> SMARTCARD handle</li> <li>• <b>TimeOutValue:</b> receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0xFFFFFFFF.</li> </ul>

Return values

- None

### 53.1.5 HAL\_SMARTCARDEx\_EnableReceiverTimeOut

Function Name      **HAL\_StatusTypeDef**  
                         **HAL\_SMARTCARDEx\_EnableReceiverTimeOut**  
                         **(SMARTCARD\_HandleTypeDef \* hsc)**

Function Description    Enable the SMARTCARD receiver timeout feature.

Parameters

- **hsc:** SMARTCARD handle

Return values

- HAL status

### 53.1.6 HAL\_SMARTCARDEx\_DisableReceiverTimeOut

Function Name      **HAL\_StatusTypeDef**  
                         **HAL\_SMARTCARDEx\_DisableReceiverTimeOut**  
                         **(SMARTCARD\_HandleTypeDef \* hsc)**

Function Description    Disable the SMARTCARD receiver timeout feature.

Parameters

- **hsc:** SMARTCARD handle

Return values

- HAL status



## 54 HAL SPDIFRX Generic Driver

### 54.1 SPDIFRX Firmware driver registers structures

#### 54.1.1 SPDIFRX\_InitTypeDef

##### Data Fields

- *uint32\_t InputSelection*
- *uint32\_t Retries*
- *uint32\_t WaitForActivity*
- *uint32\_t ChannelSelection*
- *uint32\_t DataFormat*
- *uint32\_t StereoMode*
- *uint32\_t PreambleTypeMask*
- *uint32\_t ChannelStatusMask*
- *uint32\_t ValidityBitMask*
- *uint32\_t ParityErrorMask*

##### Field Documentation

- ***uint32\_t SPDIFRX\_InitTypeDef::InputSelection***  
Specifies the SPDIF input selection. This parameter can be a value of [SPDIFRX\\_Input\\_Selection](#)
- ***uint32\_t SPDIFRX\_InitTypeDef::Retries***  
Specifies the Maximum allowed re-tries during synchronization phase. This parameter can be a value of [SPDIFRX\\_Max\\_Retries](#)
- ***uint32\_t SPDIFRX\_InitTypeDef::WaitForActivity***  
Specifies the wait for activity on SPDIF selected input. This parameter can be a value of [SPDIFRX\\_Wait\\_For\\_Activity](#).
- ***uint32\_t SPDIFRX\_InitTypeDef::ChannelSelection***  
Specifies whether the control flow will take the channel status from channel A or B. This parameter can be a value of [SPDIFRX\\_Channel\\_Selection](#)
- ***uint32\_t SPDIFRX\_InitTypeDef::DataFormat***  
Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of [SPDIFRX\\_Data\\_Format](#)
- ***uint32\_t SPDIFRX\_InitTypeDef::StereoMode***  
Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of [SPDIFRX\\_Stereo\\_Mode](#)
- ***uint32\_t SPDIFRX\_InitTypeDef::PreambleTypeMask***  
Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_PT\\_Mask](#)
- ***uint32\_t SPDIFRX\_InitTypeDef::ChannelStatusMask***  
Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_ChannelStatus\\_Mask](#)
- ***uint32\_t SPDIFRX\_InitTypeDef::ValidityBitMask***  
Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_V\\_Mask](#)
- ***uint32\_t SPDIFRX\_InitTypeDef::ParityErrorMask***  
Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_PE\\_Mask](#)

### 54.1.2 SPDIFRX\_SetDataFormatTypeDef

#### Data Fields

- *uint32\_t DataFormat*
- *uint32\_t StereoMode*
- *uint32\_t PreambleTypeMask*
- *uint32\_t ChannelStatusMask*
- *uint32\_t ValidityBitMask*
- *uint32\_t ParityErrorMask*

#### Field Documentation

- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::DataFormat*  
Specifies the Data samples format (LSB, MSB, ...). This parameter can be a value of [SPDIFRX\\_Data\\_Format](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::StereoMode*  
Specifies whether the peripheral is in stereo or mono mode. This parameter can be a value of [SPDIFRX\\_Stereo\\_Mode](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::PreambleTypeMask*  
Specifies whether The preamble type bits are copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_PT\\_Mask](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::ChannelStatusMask*  
Specifies whether the channel status and user bits are copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_ChannelStatus\\_Mask](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::ValidityBitMask*  
Specifies whether the validity bit is copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_V\\_Mask](#)
- *uint32\_t SPDIFRX\_SetDataFormatTypeDef::ParityErrorMask*  
Specifies whether the parity error bit is copied or not into the received frame. This parameter can be a value of [SPDIFRX\\_PE\\_Mask](#)

### 54.1.3 SPDIFRX\_HandleTypeDef

#### Data Fields

- *SPDIFRX\_TypeDef \* Instance*
- *SPDIFRX\_InitTypeDef Init*
- *uint32\_t \* pRxBuffPtr*
- *uint32\_t \* pCsBuffPtr*
- *\_\_IO uint16\_t RxXferSize*
- *\_\_IO uint16\_t RxXferCount*
- *\_\_IO uint16\_t CsXferSize*
- *\_\_IO uint16\_t CsXferCount*
- *DMA\_HandleTypeDef \* hdmaCsRx*
- *DMA\_HandleTypeDef \* hdmaDrRx*
- *\_\_IO HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_SPDIFRX\_StateTypeDef State*
- *\_\_IO uint32\_t ErrorCode*

## Field Documentation

- *SPDIFRX\_TypeDef\* SPDIFRX\_HandleTypeDef::Instance*
- *SPDIFRX\_InitTypeDef SPDIFRX\_HandleTypeDef::Init*
- *uint32\_t\* SPDIFRX\_HandleTypeDef::pRxBuffPtr*
- *uint32\_t\* SPDIFRX\_HandleTypeDef::pCsBuffPtr*
- *\_\_IO uint16\_t SPDIFRX\_HandleTypeDef::RxXferSize*
- *\_\_IO uint16\_t SPDIFRX\_HandleTypeDef::RxXferCount*
- *\_\_IO uint16\_t SPDIFRX\_HandleTypeDef::CsXferSize*
- *\_\_IO uint16\_t SPDIFRX\_HandleTypeDef::CsXferCount*
- *DMA\_HandleTypeDef\* SPDIFRX\_HandleTypeDef::hdmaCsRx*
- *DMA\_HandleTypeDef\* SPDIFRX\_HandleTypeDef::hdmaDrRx*
- *\_\_IO HAL\_LockTypeDef SPDIFRX\_HandleTypeDef::Lock*
- *\_\_IO HAL\_SPDIFRX\_StateTypeDef SPDIFRX\_HandleTypeDef::State*
- *\_\_IO uint32\_t SPDIFRX\_HandleTypeDef::ErrorCode*

## 54.2 SPDIFRX Firmware driver API description

### 54.2.1 How to use this driver

The SPDIFRX HAL driver can be used as follow:

1. Declare SPDIFRX\_HandleTypeDef handle structure.
2. Initialize the SPDIFRX low level resources by implement the HAL\_SPDIFRX\_MspInit() API:
  - a. Enable the SPDIFRX interface clock.
  - b. SPDIFRX pins configuration:
    - Enable the clock for the SPDIFRX GPIOs.
    - Configure these SPDIFRX pins as alternate function pull-up.
  - c. NVIC configuration if you need to use interrupt process (HAL\_SPDIFRX\_ReceiveControlFlow\_IT() and HAL\_SPDIFRX\_ReceiveDataFlow\_IT() API's).
    - Configure the SPDIFRX interrupt priority.
    - Enable the NVIC SPDIFRX IRQ handle.
  - d. DMA Configuration if you need to use DMA process (HAL\_SPDIFRX\_ReceiveDataFlow\_DMA() and HAL\_SPDIFRX\_ReceiveControlFlow\_DMA() API's).
    - Declare a DMA handle structure for the reception of the Data Flow channel.
    - Declare a DMA handle structure for the reception of the Control Flow channel.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure CtrlRx/DataRx with the required parameters.
    - Configure the DMA Channel.
    - Associate the initialized DMA handle to the SPDIFRX DMA CtrlRx/DataRx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA CtrlRx/DataRx channel.
3. Program the input selection, re-tries number, wait for activity, channel status selection, data format, stereo mode and masking of user bits using HAL\_SPDIFRX\_Init() function. The specific SPDIFRX interrupts (RXNE/CSRNE and Error Interrupts) will be managed using the macros \_\_SPDIFRX\_ENABLE\_IT() and

\_\_SPDIFRX\_DISABLE\_IT() inside the receive process. Make sure that ck\_spdif clock is configured.

4. Three operation modes are available within this driver :

### **Polling mode for reception operation (for debug purpose)**

- Receive data flow in blocking mode using HAL\_SPDIFRX\_ReceiveDataFlow()
- Receive control flow of data in blocking mode using HAL\_SPDIFRX\_ReceiveControlFlow()

### **Interrupt mode for reception operation**

- Receive an amount of data (Data Flow) in non blocking mode using HAL\_SPDIFRX\_ReceiveDataFlow\_IT()
- Receive an amount of data (Control Flow) in non blocking mode using HAL\_SPDIFRX\_ReceiveControlFlow\_IT()
- At reception end of half transfer HAL\_SPDIFRX\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_RxHalfCpltCallback
- At reception end of transfer HAL\_SPDIFRX\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_RxCpltCallback
- In case of transfer Error, HAL\_SPDIFRX\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_ErrorCallback

### **DMA mode for reception operation**

- Receive an amount of data (Data Flow) in non blocking mode (DMA) using HAL\_SPDIFRX\_ReceiveDataFlow\_DMA()
- Receive an amount of data (Control Flow) in non blocking mode (DMA) using HAL\_SPDIFRX\_ReceiveControlFlow\_DMA()
- At reception end of half transfer HAL\_SPDIFRX\_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_RxHalfCpltCallback
- At reception end of transfer HAL\_SPDIFRX\_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_RxCpltCallback
- In case of transfer Error, HAL\_SPDIFRX\_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL\_SPDIFRX\_ErrorCallback
- Stop the DMA Transfer using HAL\_SPDIFRX\_DMASStop()

### **SPDIFRX HAL driver macros list**

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

- \_\_HAL\_SPDIFRX\_IDLE: Disable the specified SPDIFRX peripheral (IDEL State)
- \_\_HAL\_SPDIFRX\_SYNC: Enable the synchronization state of the specified SPDIFRX peripheral (SYNC State)
- \_\_HAL\_SPDIFRX\_RCV: Enable the receive state of the specified SPDIFRX peripheral (RCV State)
- \_\_HAL\_SPDIFRX\_ENABLE\_IT : Enable the specified SPDIFRX interrupts
- \_\_HAL\_SPDIFRX\_DISABLE\_IT : Disable the specified SPDIFRX interrupts

- `__HAL_SPDIFRX_GET_FLAG`: Check whether the specified SPDIFRX flag is set or not.



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

### 54.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPDIFRX peripheral:

- User must Implement `HAL_SPDIFRX_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
- Call the function `HAL_SPDIFRX_Init()` to configure the SPDIFRX peripheral with the selected configuration:
  - Input Selection (IN0, IN1,...)
  - Maximum allowed re-tries during synchronization phase
  - Wait for activity on SPDIF selected input
  - Channel status selection (from channel A or B)
  - Data format (LSB, MSB, ...)
  - Stereo mode
  - User bits masking (PT,C,U,V,...)
- Call the function `HAL_SPDIFRX_DeInit()` to restore the default configuration of the selected SPDIFRXx peripheral.

This section contains the following APIs:

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

### 54.2.3 IO operation functions

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

1. There is two mode of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer start-up. The end of the data processing will be indicated through the dedicated SPDIFRX IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
  - `HAL_SPDIFRX_ReceiveDataFlow()`
  - `HAL_SPDIFRX_ReceiveControlFlow()` (+@) Do not use blocking mode to receive both control and data flow at the same time.
3. No-Blocking mode functions with Interrupt are :
  - `HAL_SPDIFRX_ReceiveControlFlow_IT()`
  - `HAL_SPDIFRX_ReceiveDataFlow_IT()`
4. No-Blocking mode functions with DMA are :

- HAL\_SPDIFRX\_ReceiveControlFlow\_DMA()
- HAL\_SPDIFRX\_ReceiveDataFlow\_DMA()
- 5. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
  - HAL\_SPDIFRX\_RxCpltCallback()
  - HAL\_SPDIFRX\_ErrorCallback()

This section contains the following APIs:

- [HAL\\_SPDIFRX\\_ReceiveDataFlow\(\)](#)
- [HAL\\_SPDIFRX\\_ReceiveControlFlow\(\)](#)
- [HAL\\_SPDIFRX\\_ReceiveDataFlow\\_IT\(\)](#)
- [HAL\\_SPDIFRX\\_ReceiveControlFlow\\_IT\(\)](#)
- [HAL\\_SPDIFRX\\_ReceiveDataFlow\\_DMA\(\)](#)
- [HAL\\_SPDIFRX\\_ReceiveControlFlow\\_DMA\(\)](#)
- [HAL\\_SPDIFRX\\_DMAStop\(\)](#)
- [HAL\\_SPDIFRX\\_IRQHandler\(\)](#)
- [HAL\\_SPDIFRX\\_RxHalfCpltCallback\(\)](#)
- [HAL\\_SPDIFRX\\_RxCpltCallback\(\)](#)
- [HAL\\_SPDIFRX\\_CxHalfCpltCallback\(\)](#)
- [HAL\\_SPDIFRX\\_CxCpltCallback\(\)](#)
- [HAL\\_SPDIFRX\\_ErrorCallback\(\)](#)

#### 54.2.4 Peripheral State and Errors functions

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

This section contains the following APIs:

- [HAL\\_SPDIFRX\\_GetState\(\)](#)
- [HAL\\_SPDIFRX\\_GetError\(\)](#)

#### 54.2.5 HAL\_SPDIFRX\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_Init</b> (SPDIFRX_HandleTypeDef * hspdif)
Function Description	Initializes the SPDIFRX according to the specified parameters in the SPDIFRX_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 54.2.6 HAL\_SPDIFRX\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_DeInit</b> (SPDIFRX_HandleTypeDef * hspdif)
Function Description	DeInitializes the SPDIFRX peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

#### 54.2.7 HAL\_SPDIFRX\_MspltInit

Function Name	<b>void HAL_SPDIFRX_MspltInit</b> (SPDIFRX_HandleTypeDef * hspdif)
Function Description	SPDIFRX MSP Init.

Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 54.2.8 HAL\_SPDIFRX\_MspDeInit

Function Name	<b>void HAL_SPDIFRX_MspDeInit (SPDIFRX_HandleTypeDef * hspdif)</b>
Function Description	SPDIFRX MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 54.2.9 HAL\_SPDIFRX\_SetDataFormat

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_SetDataFormat (SPDIFRX_HandleTypeDef * hspdif, SPDIFRX_SetDataFormatTypeDef sDataFormat)</b>
Function Description	Sets the SPDIFRX ddat format according to the specified parameters in the SPDIFRX_InitTypeDef.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> <li>• <b>sDataFormat</b>: SPDIFRX data format</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 54.2.10 HAL\_SPDIFRX\_ReceiveDataFlow

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receives an amount of data (Data Flow) in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: pointer to SPDIFRX_HandleTypeDef structure that contains the configuration information for SPDIFRX 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>

### 54.2.11 HAL\_SPDIFRX\_ReceiveControlFlow

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Receives an amount of data (Control Flow) in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: pointer to a SPDIFRX_HandleTypeDef structure that contains the configuration information for SPDIFRX 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>



**54.2.12 HAL\_SPDIFRX\_ReceiveDataFlow\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_IT</b> (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)
Function Description	Receive an amount of data (Data Flow) in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> <li>• <b>pData</b>: a 32-bit pointer to the Receive data buffer.</li> <li>• <b>Size</b>: number of data sample to be received .</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**54.2.13 HAL\_SPDIFRX\_ReceiveControlFlow\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow_IT</b> (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)
Function Description	Receive an amount of data (Control Flow) with Interrupt.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> <li>• <b>pData</b>: a 32-bit pointer to the Receive data buffer.</li> <li>• <b>Size</b>: number of data sample (Control Flow) to be received :</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**54.2.14 HAL\_SPDIFRX\_ReceiveDataFlow\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_DMA</b> (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)
Function Description	Receive an amount of data (Data Flow) mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> <li>• <b>pData</b>: a 32-bit pointer to the Receive data buffer.</li> <li>• <b>Size</b>: number of data sample to be received :</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**54.2.15 HAL\_SPDIFRX\_ReceiveControlFlow\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow_DMA</b> (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size)
Function Description	Receive an amount of data (Control Flow) with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspdif</b>: SPDIFRX handle</li> <li>• <b>pData</b>: a 32-bit pointer to the Receive data buffer.</li> <li>• <b>Size</b>: number of data (Control Flow) sample to be received :</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**54.2.16 HAL\_SPDIFRX\_DMASStop**



Function Name **HAL\_StatusTypeDef HAL\_SPDIFRX\_DMAStop (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description stop the audio stream receive from the Media.

Parameters • **hspdif**: SPDIFRX handle

Return values • None

#### 54.2.17 HAL\_SPDIFRX\_IRQHandler

Function Name **void HAL\_SPDIFRX\_IRQHandler (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description This function handles SPDIFRX interrupt request.

Parameters • **hspdif**: SPDIFRX handle

Return values • HAL status

#### 54.2.18 HAL\_SPDIFRX\_RxHalfCpltCallback

Function Name **void HAL\_SPDIFRX\_RxHalfCpltCallback (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description Rx Transfer (Data flow) half completed callbacks.

Parameters • **hspdif**: SPDIFRX handle

Return values • None

#### 54.2.19 HAL\_SPDIFRX\_RxCpltCallback

Function Name **void HAL\_SPDIFRX\_RxCpltCallback (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description Rx Transfer (Data flow) completed callbacks.

Parameters • **hspdif**: SPDIFRX handle

Return values • None

#### 54.2.20 HAL\_SPDIFRX\_CxHalfCpltCallback

Function Name **void HAL\_SPDIFRX\_CxHalfCpltCallback (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description Rx (Control flow) Transfer half completed callbacks.

Parameters • **hspdif**: SPDIFRX handle

Return values • None

#### 54.2.21 HAL\_SPDIFRX\_CxCpltCallback

Function Name **void HAL\_SPDIFRX\_CxCpltCallback (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description Rx Transfer (Control flow) completed callbacks.

Parameters • **hspdif**: SPDIFRX handle

Return values

- None

### 54.2.22 HAL\_SPDIFRX\_ErrorCallback

Function Name **void HAL\_SPDIFRX\_ErrorCallback (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description SPDIFRX error callbacks.

Parameters

- **hspdif**: SPDIFRX handle

Return values

- None

### 54.2.23 HAL\_SPDIFRX\_GetState

Function Name **HAL\_SPDIFRX\_StateTypeDef HAL\_SPDIFRX\_GetState (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description Return the SPDIFRX state.

Parameters

- **hspdif**: : SPDIFRX handle

Return values

- HAL state

### 54.2.24 HAL\_SPDIFRX\_GetError

Function Name **uint32\_t HAL\_SPDIFRX\_GetError (SPDIFRX\_HandleTypeDef \* hspdif)**

Function Description Return the SPDIFRX error code.

Parameters

- **hspdif**: : SPDIFRX handle

Return values

- SPDIFRX Error Code

## 54.3 SPDIFRX Firmware driver defines

### 54.3.1 SPDIFRX

#### ***SPDIFRX Channel Status Mask***

SPDIFRX\_CHANNELSTATUS\_OFF

SPDIFRX\_CHANNELSTATUS\_ON

#### ***SPDIFRX Channel Selection***

SPDIFRX\_CHANNEL\_A

SPDIFRX\_CHANNEL\_B

#### ***SPDIFRX Data Format***

SPDIFRX\_DATAFORMAT\_LSB

SPDIFRX\_DATAFORMAT\_MSB

SPDIFRX\_DATAFORMAT\_32BITS

#### ***SPDIFRX Error Code***

HAL\_SPDIFRX\_ERROR\_NONE      No error

HAL\_SPDIFRX\_ERROR\_TIMEOUT      Timeout error

HAL_SPDIFRX_ERROR_OVR	OVR error
HAL_SPDIFRX_ERROR_PE	Parity error
HAL_SPDIFRX_ERROR_DMA	DMA transfer error
HAL_SPDIFRX_ERROR_UNKNOWN	Unknown Error error

**SPDIFRX Exported Macros**

__HAL_SPDIFRX_RESET_HANDLE_STATE	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Reset SPDIFRX handle state.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: SPDIFRX handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_SPDIFRX_IDLE	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Disable the specified SPDIFRX peripheral (IDLE State).</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: specifies the SPDIFRX Handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_SPDIFRX_SYNC	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Enable the specified SPDIFRX peripheral (SYNC State).</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: specifies the SPDIFRX Handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_SPDIFRX_RCV	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Enable the specified SPDIFRX peripheral (RCV State).</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>__HANDLE__: specifies the SPDIFRX Handle.</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None</li> </ul>
__HAL_SPDIFRX_ENABLE_IT	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Enable or disable the specified SPDIFRX interrupts.</li> </ul>

`__HAL_SPDIFRX_DISABLE_IT`  
`__HAL_SPDIFRX_GET_IT_SOURCE`

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.
- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - `SPDIFRX_IT_RXNE`
  - `SPDIFRX_IT_CSRNE`
  - `SPDIFRX_IT_PERRIE`
  - `SPDIFRX_IT_OVRIE`
  - `SPDIFRX_IT_SBLKIE`
  - `SPDIFRX_IT_SYNCDIE`
  - `SPDIFRX_IT_IFEIE`

**Return value:**

- None

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.
- `__INTERRUPT__`: specifies the SPDIFRX interrupt source to check. This parameter can be one of the following values:
  - `SPDIFRX_IT_RXNE`
  - `SPDIFRX_IT_CSRNE`
  - `SPDIFRX_IT_PERRIE`
  - `SPDIFRX_IT_OVRIE`
  - `SPDIFRX_IT_SBLKIE`
  - `SPDIFRX_IT_SYNCDIE`
  - `SPDIFRX_IT_IFEIE`

**Return value:**

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

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the SPDIFRX Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:

`__HAL_SPDIFRX_GET_FLAG`

- SPDIFRX\_FLAG\_RXNE
- SPDIFRX\_FLAG\_CSRNE
- SPDIFRX\_FLAG\_PERR
- SPDIFRX\_FLAG\_OVR
- SPDIFRX\_FLAG\_SBD
- SPDIFRX\_FLAG\_SYNCD
- SPDIFRX\_FLAG\_FERR
- SPDIFRX\_FLAG\_SERR
- SPDIFRX\_FLAG\_TERR

**Return value:**

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

**Description:**

- Clears the specified SPDIFRX SR flag, in setting the proper IFCR register bit.

**Parameters:**

- \_\_HANDLE\_\_: specifies the USART Handle.
- \_\_IT\_CLEAR\_\_: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - SPDIFRX\_FLAG\_PERR
  - SPDIFRX\_FLAG\_OVR
  - SPDIFRX\_SR\_SBD
  - SPDIFRX\_SR\_SYNCD

**Return value:**

- None

\_\_HAL\_SPDIFRX\_CLEAR\_IT

***SPDIFRX Flags Definition***

SPDIFRX\_FLAG\_RXNE

SPDIFRX\_FLAG\_CSRNE

SPDIFRX\_FLAG\_PERR

SPDIFRX\_FLAG\_OVR

SPDIFRX\_FLAG\_SBD

SPDIFRX\_FLAG\_SYNCD

SPDIFRX\_FLAG\_FERR

SPDIFRX\_FLAG\_SERR

SPDIFRX\_FLAG\_TERR

***SPDIFRX Input Selection***

SPDIFRX\_INPUT\_IN0

SPDIFRX\_INPUT\_IN1

SPDIFRX\_INPUT\_IN2

SPDIFRX\_INPUT\_IN3

***SPDIFRX Interrupts Definition***

SPDIFRX\_IT\_RXNE

SPDIFRX\_IT\_CSRNE

SPDIFRX\_IT\_PERRIE

SPDIFRX\_IT\_OVRIE

SPDIFRX\_IT\_SBLKIE

SPDIFRX\_IT\_SYNCDIE

SPDIFRX\_IT\_IFEIE

***SPDIFRX Maximum Retries***

SPDIFRX\_MAXRETRIES\_NONE

SPDIFRX\_MAXRETRIES\_3

SPDIFRX\_MAXRETRIES\_15

SPDIFRX\_MAXRETRIES\_63

***SPDIFRX Parity Error Mask***

SPDIFRX\_PARITYERRORMASK\_OFF

SPDIFRX\_PARITYERRORMASK\_ON

***SPDIFRX Private Macros***

IS\_SPDIFRX\_INPUT\_SELECT

IS\_SPDIFRX\_MAX\_RETRIES

IS\_SPDIFRX\_WAIT\_FOR\_ACTIVITY

IS\_PREAMBLE\_TYPE\_MASK

IS\_VALIDITY\_MASK

IS\_PARITY\_ERROR\_MASK

IS\_SPDIFRX\_CHANNEL

IS\_SPDIFRX\_DATA\_FORMAT

IS\_STEREO\_MODE

IS\_CHANNEL\_STATUS\_MASK

***SPDIFRX Preamble Type Mask***

SPDIFRX\_PREAMBLETYPEMASK\_OFF

SPDIFRX\_PREAMBLETYPEMASK\_ON

***SPDIFRX State***

SPDIFRX\_STATE\_IDLE

SPDIFRX\_STATE\_SYNC

SPDIFRX\_STATE\_RCV

***SPDIFRX Stereo Mode***

SPDIFRX\_STEREOMODE\_DISABLE

SPDIFRX\_STEREOMODE\_ENABLE

***SPDIFRX Validity Mask***

SPDIFRX\_VALIDITYMASK\_OFF

SPDIFRX\_VALIDITYMASK\_ON

***SPDIFRX Wait For Activity***

SPDIFRX\_WAITFORACTIVITY\_OFF

SPDIFRX\_WAITFORACTIVITY\_ON

## 55 HAL SPI Generic Driver

### 55.1 SPI Firmware driver registers structures

#### 55.1.1 SPI\_InitTypeDef

##### Data Fields

- *uint32\_t Mode*
- *uint32\_t Direction*
- *uint32\_t DataSize*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t NSS*
- *uint32\_t BaudRatePrescaler*
- *uint32\_t FirstBit*
- *uint32\_t TIMode*
- *uint32\_t CRCCalculation*
- *uint32\_t CRCPolynomial*
- *uint32\_t CRCLength*
- *uint32\_t NSSPMode*

##### Field Documentation

- ***uint32\_t SPI\_InitTypeDef::Mode***  
Specifies the SPI operating mode. This parameter can be a value of [SPI\\_Mode](#)
- ***uint32\_t SPI\_InitTypeDef::Direction***  
Specifies the SPI bidirectional mode state. This parameter can be a value of [SPI\\_Direction](#)
- ***uint32\_t SPI\_InitTypeDef::DataSize***  
Specifies the SPI data size. This parameter can be a value of [SPI\\_Data\\_Size](#)
- ***uint32\_t SPI\_InitTypeDef::CLKPolarity***  
Specifies the serial clock steady state. This parameter can be a value of [SPI\\_Clock\\_Polarity](#)
- ***uint32\_t SPI\_InitTypeDef::CLKPhase***  
Specifies the clock active edge for the bit capture. This parameter can be a value of [SPI\\_Clock\\_Phase](#)
- ***uint32\_t SPI\_InitTypeDef::NSS***  
Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of [SPI\\_Slave\\_Select\\_management](#)
- ***uint32\_t SPI\_InitTypeDef::BaudRatePrescaler***  
Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of [SPI\\_BaudRate\\_Prescaler](#)  
**Note:** The communication clock is derived from the master clock. The slave clock does not need to be set.
- ***uint32\_t SPI\_InitTypeDef::FirstBit***  
Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [SPI\\_MSB\\_LSB\\_transmission](#)



- ***uint32\_t SPI\_InitTypeDef::TMode***  
Specifies if the TI mode is enabled or not . This parameter can be a value of [SPI\\_TI\\_mode](#)
- ***uint32\_t SPI\_InitTypeDef::CRCCalculation***  
Specifies if the CRC calculation is enabled or not. This parameter can be a value of [SPI\\_CRC\\_Calculation](#)
- ***uint32\_t SPI\_InitTypeDef::CRCPolynomial***  
Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min\_Data = 0 and Max\_Data = 65535
- ***uint32\_t SPI\_InitTypeDef::CRCLength***  
Specifies the CRC Length used for the CRC calculation. CRC Length is only used with Data8 and Data16, not other data size This parameter can be a value of [SPI\\_CRC\\_length](#)
- ***uint32\_t SPI\_InitTypeDef::NSSPMode***  
Specifies whether the NSSP signal is enabled or not . This parameter can be a value of [SPI\\_NSSP\\_Mode](#) This mode is activated by the NSSP bit in the SPIx\_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx\_CR1 CPHA = 0, CPOL setting is ignored)..

### 55.1.2 \_\_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***
- ***uint32\_t CRCSize***
- ***void(\* RxISR***
- ***void(\* TxISR***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***HAL\_SPI\_StateTypeDef State***
- ***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***

- `uint32_t __SPI_HandleTypeDef::CRCSize`
- `void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)`
- `void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)`
- `DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx`
- `HAL_LockTypeDef __SPI_HandleTypeDef::Lock`
- `HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State`
- `uint32_t __SPI_HandleTypeDef::ErrorCode`

## 55.2 SPI Firmware driver API description

### 55.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI\_HandleTypeDef handle structure, for example: SPI\_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL\_SPI\_MspInit ()API:
  - a. Enable the SPIx interface clock
  - b. SPI pins configuration
    - Enable the clock for the SPI GPIOs
    - Configure these SPI pins as alternate function push-pull
  - c. NVIC configuration if you need to use interrupt process
    - Configure the SPIx interrupt priority
    - Enable the NVIC SPI IRQ handle
  - d. DMA Configuration if you need to use DMA process
    - Declare a DMA\_HandleTypeDef handle structure for the transmit or receive channel
    - Enable the DMAx clock
    - Configure the DMA handle parameters
    - Configure the DMA Tx or Rx channel
    - Associate the 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 channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL\_SPI\_Init() API:
  - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customised 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

### 55.2.2 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
  - TMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
  - CRC Length, used only with Data8 and Data16
  - FIFO reception threshold
- Call the function HAL\_SPI\_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

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

### 55.2.3 IO operation functions

The SPI supports master and slave mode :

1. There are two modes of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_SPI\_TxCpltCallback(), HAL\_SPI\_RxCpltCallback() and HAL\_SPI\_TxRxCpltCallback() user callbacks will be executed 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\\_DMABuffer\(\)](#)

- [HAL\\_SPI\\_DMAResume\(\)](#)
- [HAL\\_SPI\\_DMAStop\(\)](#)
- [HAL\\_SPI\\_IRQHandler\(\)](#)
- [HAL\\_SPI\\_TxCpltCallback\(\)](#)
- [HAL\\_SPI\\_RxCpltCallback\(\)](#)
- [HAL\\_SPI\\_TxRxCpltCallback\(\)](#)
- [HAL\\_SPI\\_TxHalfCpltCallback\(\)](#)
- [HAL\\_SPI\\_RxHalfCpltCallback\(\)](#)
- [HAL\\_SPI\\_TxRxHalfCpltCallback\(\)](#)
- [HAL\\_SPI\\_ErrorCallback\(\)](#)

### 55.2.4 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\(\)](#)

### 55.2.5 HAL\_SPI\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)</b>
Function Description	Initializes the SPI according to the specified parameters in the SPI_InitTypeDef and create the associated handle.
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 status</li> </ul>

### 55.2.6 HAL\_SPI\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)</b>
Function Description	DeInitializes the SPI peripheral.
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 status</li> </ul>

### 55.2.7 HAL\_SPI\_MspInit

Function Name	<b>void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI MSP Init.
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>

### 55.2.8 HAL\_SPI\_MspDeInit

Function Name	<b>void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI MSP DeInit.
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>

### 55.2.9 HAL\_SPI\_Transmit

Function Name	<b>HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)</b>
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li><b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li> <li><b>pData:</b> pointer to data buffer</li> <li><b>Size:</b> amount of data to be sent</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 55.2.10 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 received</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 55.2.11 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</li> <li><b>Size:</b> amount of data to be sent and received</li> <li><b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 55.2.12 HAL\_SPI\_Transmit\_IT

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

---

**(SPI\_HandleTypeDef \* hspi, uint8\_t \* pData, uint16\_t Size)**

Function Description Transmit an amount of data in no-blocking mode with Interrupt.

Parameters

- **hspi**: pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData**: pointer to data buffer
- **Size**: amount of data to be sent

Return values

- HAL status

### 55.2.13 HAL\_SPI\_Receive\_IT

Function Name **HAL\_StatusTypeDef HAL\_SPI\_Receive\_IT**  
**(SPI\_HandleTypeDef \* hspi, uint8\_t \* pData, uint16\_t Size)**

Function Description Receive an amount of data in no-blocking mode with Interrupt.

Parameters

- **hspi**: pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData**: pointer to data buffer
- **Size**: amount of data to be sent

Return values

- HAL status

### 55.2.14 HAL\_SPI\_TransmitReceive\_IT

Function Name **HAL\_StatusTypeDef HAL\_SPI\_TransmitReceive\_IT**  
**(SPI\_HandleTypeDef \* hspi, uint8\_t \* pTxData, uint8\_t \* pRxData, uint16\_t Size)**

Function Description Transmit and Receive an amount of data in no-blocking mode with Interrupt.

Parameters

- **hspi**: pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData**: pointer to transmission data buffer
- **pRxData**: pointer to reception data buffer
- **Size**: amount of data to be sent and received

Return values

- HAL status

### 55.2.15 HAL\_SPI\_Transmit\_DMA

Function Name **HAL\_StatusTypeDef HAL\_SPI\_Transmit\_DMA**  
**(SPI\_HandleTypeDef \* hspi, uint8\_t \* pData, uint16\_t Size)**

Function Description Transmit an amount of data in no-blocking mode with DMA.

Parameters

- **hspi**: pointer to a SPI\_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData**: pointer to data buffer
- **Size**: amount of data to be sent

Return values

- HAL status

### 55.2.16 HAL\_SPI\_Receive\_DMA

Function Name **HAL\_StatusTypeDef HAL\_SPI\_Receive\_DMA**

---

(SPI\_HandleTypeDef \* hspi, uint8\_t \* pData, uint16\_t Size)

Function Description	Receive an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>hspi</b>: SPI handle</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>

### 55.2.17 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>

### 55.2.18 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>

### 55.2.19 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>

### 55.2.20 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>

### 55.2.21 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 the configuration information for the specified SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 55.2.22 HAL\_SPI\_TxCpltCallback

Function Name	<b>void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx Transfer completed callback.
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>

### 55.2.23 HAL\_SPI\_RxCpltCallback

Function Name	<b>void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>hspi:</b> pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 55.2.24 HAL\_SPI\_TxRxCpltCallback

Function Name	<b>void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx and Rx Transfer completed callback.
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>

### 55.2.25 HAL\_SPI\_TxHalfCpltCallback

Function Name	<b>void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx Half Transfer completed callback.
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>

### 55.2.26 HAL\_SPI\_RxHalfCpltCallback



Function Name	<b>void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Rx Half Transfer completed callback.
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>

### 55.2.27 HAL\_SPI\_TxRxHalfCpltCallback

Function Name	<b>void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	Tx and Rx Half Transfer callback.
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>

### 55.2.28 HAL\_SPI\_ErrorCallback

Function Name	<b>void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)</b>
Function Description	SPI error callback.
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>

### 55.2.29 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>SPI state</li> </ul>

### 55.2.30 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 in bitmap format</li> </ul>

## 55.3 SPI Firmware driver defines

### 55.3.1 SPI

*SPI BaudRate Prescaler*

SPI\_BAUDRATEPRESCALER\_2  
SPI\_BAUDRATEPRESCALER\_4  
SPI\_BAUDRATEPRESCALER\_8  
SPI\_BAUDRATEPRESCALER\_16  
SPI\_BAUDRATEPRESCALER\_32  
SPI\_BAUDRATEPRESCALER\_64  
SPI\_BAUDRATEPRESCALER\_128  
SPI\_BAUDRATEPRESCALER\_256

***SPI Clock Phase***

SPI\_PHASE\_1EDGE  
SPI\_PHASE\_2EDGE

***SPI Clock Polarity***

SPI\_POLARITY\_LOW  
SPI\_POLARITY\_HIGH

***SPI CRC Calculation***

SPI\_CRCCALCULATION\_DISABLE  
SPI\_CRCCALCULATION\_ENABLE

***SPI CRC Length***

SPI\_CRC\_LENGTH\_DATASIZE  
SPI\_CRC\_LENGTH\_8BIT  
SPI\_CRC\_LENGTH\_16BIT

***SPI Data Size***

SPI\_DATASIZE\_4BIT  
SPI\_DATASIZE\_5BIT  
SPI\_DATASIZE\_6BIT  
SPI\_DATASIZE\_7BIT  
SPI\_DATASIZE\_8BIT  
SPI\_DATASIZE\_9BIT  
SPI\_DATASIZE\_10BIT  
SPI\_DATASIZE\_11BIT  
SPI\_DATASIZE\_12BIT  
SPI\_DATASIZE\_13BIT  
SPI\_DATASIZE\_14BIT  
SPI\_DATASIZE\_15BIT  
SPI\_DATASIZE\_16BIT

***SPI 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	Error on BSY/TXE/FTLVL/FRLVL Flag
HAL_SPI_ERROR_UNKNOW	Unknow Error error

### ***SPI Exported Macros***

`__HAL_SPI_RESET_HANDLE_STATE` **Description:**

- Reset SPI handle state.

#### **Parameters:**

- `__HANDLE__`: SPI handle.

#### **Return value:**

- None

`__HAL_SPI_ENABLE_IT`

#### **Description:**

- Enables or disables the specified SPI interrupts.

#### **Parameters:**

- `__HANDLE__`: specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__INTERRUPT__`: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - SPI\_IT\_TXE: Tx buffer empty interrupt enable
  - SPI\_IT\_RXNE: RX buffer not empty interrupt enable
  - SPI\_IT\_ERR: Error interrupt enable

#### **Return value:**

- None

`__HAL_SPI_DISABLE_IT`

`__HAL_SPI_GET_IT_SOURCE`

#### **Description:**

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

`__HAL_SPI_GET_FLAG`**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).

**Description:**

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

**Parameters:**

- `__HANDLE__` : specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__FLAG__` : specifies the flag to check.  
This parameter can be one of the following values:
  - `SPI_FLAG_RXNE`: Receive buffer not empty flag
  - `SPI_FLAG_TXE`: Transmit buffer empty flag
  - `SPI_FLAG_CRCERR`: CRC error flag
  - `SPI_FLAG_MODF`: Mode fault flag
  - `SPI_FLAG_OVR`: Overrun flag
  - `SPI_FLAG_BSY`: Busy flag
  - `SPI_FLAG_FRE`: Frame format error flag
  - `SPI_FLAG_FTLVL`: SPI fifo transmission level
  - `SPI_FLAG_FRLVL`: SPI fifo reception level

**Return value:**

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

`__HAL_SPI_CLEAR_CRCERRFLAG`**Description:**

- Clears the SPI CRCERR pending flag.

**Parameters:**

- `__HANDLE__` : specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**\_\_HAL\_SPI\_CLEAR\_MODFFLAG****Return value:**

- None

**Description:**

- Clears the SPI MODF pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: : specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

**Description:**

- Clears the SPI OVR pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: : specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

**Description:**

- Clears the SPI FRE pending flag.

**Parameters:**

- **\_\_HANDLE\_\_**: : specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

**Description:**

- Enables the SPI.

**Parameters:**

- **\_\_HANDLE\_\_**: : specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

**Description:**

- Disables the SPI.

**Parameters:**

- **\_\_HANDLE\_\_**: : specifies the SPI Handle.  
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**\_\_HAL\_SPI\_CLEAR\_OVRFLAG****\_\_HAL\_SPI\_CLEAR\_FREFLAG****\_\_HAL\_SPI\_ENABLE****\_\_HAL\_SPI\_DISABLE**

3 to select the SPI peripheral.

**Return value:**

- None

***SPI FIFO Reception Threshold***

SPI\_RXFIFO\_THRESHOLD

SPI\_RXFIFO\_THRESHOLD\_QF

SPI\_RXFIFO\_THRESHOLD\_HF

***SPI Flag definition***

SPI\_FLAG\_RXNE

SPI\_FLAG\_TXE

SPI\_FLAG\_BSY

SPI\_FLAG\_CRCERR

SPI\_FLAG\_MODF

SPI\_FLAG\_OVR

SPI\_FLAG\_FRE

SPI\_FLAG\_FTLVL

SPI\_FLAG\_FRLVL

***SPI Interrupt configuration definition***

SPI\_IT\_TXE

SPI\_IT\_RXNE

SPI\_IT\_ERR

***SPI Mode***

SPI\_MODE\_SLAVE

SPI\_MODE\_MASTER

***SPI MSB LSB transmission***

SPI\_FIRSTBIT\_MSB

SPI\_FIRSTBIT\_LSB

***SPI NSS Pulse Mode***

SPI\_NSS\_PULSE\_ENABLE

SPI\_NSS\_PULSE\_DISABLE

***SPI Private Constants***

SPI\_DEFAULT\_TIMEOUT

***SPI Private Macros***

SPI\_1LINE\_TX

**Description:**

- Sets the SPI transmit-only mode.

**Parameters:**

- `__HANDLE__`: specifies the SPI Handle.

This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

**Description:**

- Sets the SPI receive-only mode.

**Parameters:**

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

**Description:**

- Resets the CRC calculation of the SPI.

**Parameters:**

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

**Return value:**

- None

SPI\_1LINE\_RX

SPI\_RESET\_CRC

IS\_SPI\_MODE

IS\_SPI\_DIRECTION

IS\_SPI\_DIRECTION\_2LINES

IS\_SPI\_DIRECTION\_2LINES\_OR\_1LINE

IS\_SPI\_DATASIZE

IS\_SPI\_CPOL

IS\_SPI\_CPHA

IS\_SPI\_NSS

IS\_SPI\_NSSP

IS\_SPI\_BAUDRATE\_PRESCALER

IS\_SPI\_FIRST\_BIT

IS\_SPI\_TIMODE

IS\_SPI\_CRC\_CALCULATION

IS\_SPI\_CRC\_LENGTH

IS\_SPI\_CRC\_POLYNOMIAL

***SPI Reception FIFO Status Level***

SPI\_FRLVL\_EMPTY

SPI\_FRLVL\_QUARTER\_FULL

SPI\_FRLVL\_HALF\_FULL

SPI\_FRLVL\_FULL

***SPI Slave Select management***

SPI\_NSS\_SOFT

SPI\_NSS\_HARD\_INPUT

SPI\_NSS\_HARD\_OUTPUT

***SPI TI mode***

SPI\_TIMODE\_DISABLE

SPI\_TIMODE\_ENABLE

***SPI Transmission FIFO Status Level***

SPI\_FTLVL\_EMPTY

SPI\_FTLVL\_QUARTER\_FULL

SPI\_FTLVL\_HALF\_FULL

SPI\_FTLVL\_FULL



## 56 HAL SRAM Generic Driver

### 56.1 SRAM Firmware driver registers structures

#### 56.1.1 SRAM\_HandleTypeDef

##### Data Fields

- *FMC\_NORSRAM\_TypeDef \* Instance*
- *FMC\_NORSRAM\_EXTENDED\_TypeDef \* Extended*
- *FMC\_NORSRAM\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_SRAM\_StateTypeDef State*
- *DMA\_HandleTypeDef \* hdma*

##### Field Documentation

- *FMC\_NORSRAM\_TypeDef\* SRAM\_HandleTypeDef::Instance*  
Register base address
- *FMC\_NORSRAM\_EXTENDED\_TypeDef\* SRAM\_HandleTypeDef::Extended*  
Extended mode register base address
- *FMC\_NORSRAM\_InitTypeDef SRAM\_HandleTypeDef::Init*  
SRAM device control configuration parameters
- *HAL\_LockTypeDef SRAM\_HandleTypeDef::Lock*  
SRAM locking object
- *\_\_IO HAL\_SRAM\_StateTypeDef SRAM\_HandleTypeDef::State*  
SRAM device access state
- *DMA\_HandleTypeDef\* SRAM\_HandleTypeDef::hdma*  
Pointer DMA handler

### 56.2 SRAM Firmware driver API description

#### 56.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM\_HandleTypeDef handle structure, for example: SRAM\_HandleTypeDef hsram; and:
  - Fill the SRAM\_HandleTypeDef handle "Init" field with the allowed values of the structure member.
  - Fill the SRAM\_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
  - Fill the SRAM\_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FMC\_NORSRAM\_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC\_NORSRAM\_TimingTypeDef Timing and

- FMC\_NORSRAM\_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL\_SRAM\_Init(). This function performs the following sequence:
    - a. MSP hardware layer configuration using the function HAL\_SRAM\_MspInit()
    - b. Control register configuration using the FMC NORSRAM interface function FMC\_NORSRAM\_Init()
    - c. Timing register configuration using the FMC NORSRAM interface function FMC\_NORSRAM\_Timing\_Init()
    - d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC\_NORSRAM\_Extended\_Timing\_Init()
    - e. Enable the SRAM device using the macro \_\_FMC\_NORSRAM\_ENABLE()
  4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
    - HAL\_SRAM\_Read()/HAL\_SRAM\_Write() for polling read/write access
    - HAL\_SRAM\_Read\_DMA()/HAL\_SRAM\_Write\_DMA() for DMA read/write transfer
  5. You can also control the SRAM device by calling the control APIs HAL\_SRAM\_WriteOperation\_Enable()/ HAL\_SRAM\_WriteOperation\_Disable() to respectively enable/disable the SRAM write operation
  6. You can continuously monitor the SRAM device HAL state by calling the function HAL\_SRAM\_GetState()

### 56.2.2 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\(\)](#)

### 56.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

This section contains the following APIs:

- [HAL\\_SRAM\\_Read\\_8b\(\)](#)
- [HAL\\_SRAM\\_Write\\_8b\(\)](#)
- [HAL\\_SRAM\\_Read\\_16b\(\)](#)
- [HAL\\_SRAM\\_Write\\_16b\(\)](#)
- [HAL\\_SRAM\\_Read\\_32b\(\)](#)
- [HAL\\_SRAM\\_Write\\_32b\(\)](#)
- [HAL\\_SRAM\\_Read\\_DMA\(\)](#)
- [HAL\\_SRAM\\_Write\\_DMA\(\)](#)
- [HAL\\_SRAM\\_DMA\\_XferCpltCallback\(\)](#)
- [HAL\\_SRAM\\_DMA\\_XferErrorCallback\(\)](#)

### 56.2.4 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\(\)](#)

## 56.2.5 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\(\)](#)

## 56.2.6 HAL\_SRAM\_Init

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)</b>
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>

## 56.2.7 HAL\_SRAM\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)</b>
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>

## 56.2.8 HAL\_SRAM\_MspInit

Function Name	<b>void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)</b>
Function Description	SRAM MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

## 56.2.9 HAL\_SRAM\_MspDeInit

Function Name	<b>void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)</b>
Function Description	SRAM MSP DeInit.
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>

**56.2.10 HAL\_SRAM\_DMA\_XferCpltCallback**

Function Name	<b>void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)</b>
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>

**56.2.11 HAL\_SRAM\_DMA\_XferErrorCallback**

Function Name	<b>void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)</b>
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>

**56.2.12 HAL\_SRAM\_Read\_8b**

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)</b>
Function Description	Reads 8-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"><li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li><li>• <b>pAddress</b>: Pointer to read start address</li><li>• <b>pDstBuffer</b>: Pointer to destination buffer</li><li>• <b>BufferSize</b>: Size of the buffer to read from memory</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

**56.2.13 HAL\_SRAM\_Write\_8b**

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)</b>
Function Description	Writes 8-bit buffer to SRAM memory.
Parameters	<ul style="list-style-type: none"><li>• <b>hsram</b>: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li><li>• <b>pAddress</b>: Pointer to write start address</li><li>• <b>pSrcBuffer</b>: Pointer to source buffer to write</li><li>• <b>BufferSize</b>: Size of the buffer to write to memory</li></ul>
Return values	<ul style="list-style-type: none"><li>• HAL status</li></ul>

**56.2.14 HAL\_SRAM\_Read\_16b**

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t</b>
---------------	---

\* **pDstBuffer**, **uint32\_t BufferSize**)

Function Description Reads 16-bit buffer from SRAM memory.

Parameters

- **hsram**: pointer to a **SRAM\_HandleTypeDef** structure that contains the configuration information for SRAM module.
- **pAddress**: Pointer to read start address
- **pDstBuffer**: Pointer to destination buffer
- **BufferSize**: Size of the buffer to read from memory

Return values

- HAL status

### 56.2.15 HAL\_SRAM\_Write\_16b

Function Name **HAL\_StatusTypeDef HAL\_SRAM\_Write\_16b**  
(**SRAM\_HandleTypeDef \* hsram**, **uint32\_t \* pAddress**, **uint16\_t \* pSrcBuffer**, **uint32\_t BufferSize**)

Function Description Writes 16-bit buffer to SRAM memory.

Parameters

- **hsram**: pointer to a **SRAM\_HandleTypeDef** structure that contains the configuration information for SRAM module.
- **pAddress**: Pointer to write start address
- **pSrcBuffer**: Pointer to source buffer to write
- **BufferSize**: Size of the buffer to write to memory

Return values

- HAL status

### 56.2.16 HAL\_SRAM\_Read\_32b

Function Name **HAL\_StatusTypeDef HAL\_SRAM\_Read\_32b**  
(**SRAM\_HandleTypeDef \* hsram**, **uint32\_t \* pAddress**, **uint32\_t \* pDstBuffer**, **uint32\_t BufferSize**)

Function Description Reads 32-bit buffer from SRAM memory.

Parameters

- **hsram**: pointer to a **SRAM\_HandleTypeDef** structure that contains the configuration information for SRAM module.
- **pAddress**: Pointer to read start address
- **pDstBuffer**: Pointer to destination buffer
- **BufferSize**: Size of the buffer to read from memory

Return values

- HAL status

### 56.2.17 HAL\_SRAM\_Write\_32b

Function Name **HAL\_StatusTypeDef HAL\_SRAM\_Write\_32b**  
(**SRAM\_HandleTypeDef \* hsram**, **uint32\_t \* pAddress**, **uint32\_t \* pSrcBuffer**, **uint32\_t BufferSize**)

Function Description Writes 32-bit buffer to SRAM memory.

Parameters

- **hsram**: pointer to a **SRAM\_HandleTypeDef** structure that contains the configuration information for SRAM module.
- **pAddress**: Pointer to write start address
- **pSrcBuffer**: Pointer to source buffer to write
- **BufferSize**: Size of the buffer to write to memory

Return values

- HAL status

**56.2.18 HAL\_SRAM\_Read\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Read_DMA</b> (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)
Function Description	Reads a Words data from the SRAM memory using DMA transfer.
Parameters	<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>

**56.2.19 HAL\_SRAM\_Write\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_Write_DMA</b> (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)
Function Description	Writes a Words data buffer to SRAM memory using DMA transfer.
Parameters	<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>

**56.2.20 HAL\_SRAM\_DMA\_XferCpltCallback**

Function Name	<b>void HAL_SRAM_DMA_XferCpltCallback</b> (DMA_HandleTypeDef * hdma)
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>

**56.2.21 HAL\_SRAM\_DMA\_XferErrorCallback**

Function Name	<b>void HAL_SRAM_DMA_XferErrorCallback</b> (DMA_HandleTypeDef * hdma)
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>

**56.2.22 HAL\_SRAM\_WriteOperation\_Enable**

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable</b> (SRAM_HandleTypeDef * hsram)
---------------	---

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>

### 56.2.23 HAL\_SRAM\_WriteOperation\_Disable

Function Name	<b>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)</b>
Function Description	Disables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"> <li><b>hsram:</b> pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

### 56.2.24 HAL\_SRAM\_GetState

Function Name	<b>HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)</b>
Function Description	Returns the SRAM controller state.
Parameters	<ul style="list-style-type: none"> <li><b>hsram:</b> pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

## 56.3 SRAM Firmware driver defines

### 56.3.1 SRAM

#### *SRAM Exported Macros*

<b>__HAL_SRAM_RESET_HANDLE_STATE</b>	<b>Description:</b> <ul style="list-style-type: none"> <li>Reset SRAM handle state.</li> </ul> <b>Parameters:</b> <ul style="list-style-type: none"> <li><b>__HANDLE__:</b> SRAM handle</li> </ul> <b>Return value:</b> <ul style="list-style-type: none"> <li>None</li> </ul>
--------------------------------------	--

## 57 HAL TIM Generic Driver

### 57.1 TIM Firmware driver registers structures

#### 57.1.1 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.

#### 57.1.2 TIM\_OC\_InitTypeDef

##### Data Fields

- *uint32\_t OCMode*
- *uint32\_t Pulse*
- *uint32\_t OCPolarity*
- *uint32\_t OCNPolarity*
- *uint32\_t OCFastMode*
- *uint32\_t OCIdleState*
- *uint32\_t OCNIdleState*



## Field Documentation

- ***uint32\_t TIM\_OC\_InitTypeDef::OCMode***  
Specifies the TIM mode. This parameter can be a value of [TIMEx\\_Output\\_Compare\\_and\\_PWM\\_modes](#)
- ***uint32\_t TIM\_OC\_InitTypeDef::Pulse***  
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF
- ***uint32\_t TIM\_OC\_InitTypeDef::OCPolarity***  
Specifies the output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_Polarity](#)
- ***uint32\_t TIM\_OC\_InitTypeDef::OCNPolarity***  
Specifies the complementary output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Polarity](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OC\_InitTypeDef::OCFastMode***  
Specifies the Fast mode state. This parameter can be a value of [TIM\\_Output\\_Fast\\_State](#)  
**Note:**This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32\_t TIM\_OC\_InitTypeDef::OCIdleState***  
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [TIM\\_Output\\_Compare\\_Idle\\_State](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OC\_InitTypeDef::OCNIdleState***  
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Idle\\_State](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.

## 57.1.3 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 ICPolarity***
- ***uint32\_t ICSelection***
- ***uint32\_t ICFILTER***

## Field Documentation

- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCMode***  
Specifies the TIM mode. This parameter can be a value of [TIMEx\\_Output\\_Compare\\_and\\_PWM\\_modes](#)
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::Pulse***  
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF

- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCPolarity***  
Specifies the output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_Polarity](#)
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCNPolarity***  
Specifies the complementary output polarity. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Polarity](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCIdleState***  
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [TIM\\_Output\\_Compare\\_Idle\\_State](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::OCNIdleState***  
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [TIM\\_Output\\_Compare\\_N\\_Idle\\_State](#)  
**Note:**This parameter is valid only for TIM1 and TIM8.
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::ICPolarity***  
Specifies the active edge of the input signal. This parameter can be a value of [TIM\\_Input\\_Capture\\_Polarity](#)
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::ICSelection***  
Specifies the input. This parameter can be a value of [TIM\\_Input\\_Capture\\_Selection](#)
- ***uint32\_t TIM\_OnePulse\_InitTypeDef::ICFilter***  
Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

#### 57.1.4 TIM\_IC\_InitTypeDef

##### Data Fields

- ***uint32\_t ICPolarity***
- ***uint32\_t ICSelection***
- ***uint32\_t ICPrescaler***
- ***uint32\_t ICFilter***

##### Field Documentation

- ***uint32\_t TIM\_IC\_InitTypeDef::ICPolarity***  
Specifies the active edge of the input signal. This parameter can be a value of [TIM\\_Input\\_Capture\\_Polarity](#)
- ***uint32\_t TIM\_IC\_InitTypeDef::ICSelection***  
Specifies the input. This parameter can be a value of [TIM\\_Input\\_Capture\\_Selection](#)
- ***uint32\_t TIM\_IC\_InitTypeDef::ICPrescaler***  
Specifies the Input Capture Prescaler. This parameter can be a value of [TIM\\_Input\\_Capture\\_Prescaler](#)
- ***uint32\_t TIM\_IC\_InitTypeDef::ICFilter***  
Specifies the input capture filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

#### 57.1.5 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

**57.1.6 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

### 57.1.7 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 [TIMEx\\_ClearInput\\_Source](#)
- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputPolarity***  
TIM Clear Input polarity. This parameter can be a value of [TIM\\_ClearInput\\_Polarity](#)
- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputPrescaler***  
TIM Clear Input prescaler. This parameter can be a value of [TIM\\_ClearInput\\_Prescaler](#)
- ***uint32\_t TIM\_ClearInputConfigTypeDef::ClearInputFilter***  
TIM Clear Input filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

### 57.1.8 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 [TIMEx\\_Slave\\_Mode](#)
- ***uint32\_t TIM\_SlaveConfigTypeDef::InputTrigger***  
Input Trigger source This parameter can be a value of [TIM\\_Trigger\\_Selection](#)
- ***uint32\_t TIM\_SlaveConfigTypeDef::TriggerPolarity***  
Input Trigger polarity This parameter can be a value of [TIM\\_Trigger\\_Polarity](#)
- ***uint32\_t TIM\_SlaveConfigTypeDef::TriggerPrescaler***  
Input trigger prescaler This parameter can be a value of [TIM\\_Trigger\\_Prescaler](#)
- ***uint32\_t TIM\_SlaveConfigTypeDef::TriggerFilter***  
Input trigger filter This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF

### 57.1.9 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

## 57.2 TIM Firmware driver API description

### 57.2.1 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

## 57.2.2 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()`

## 57.2.3 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\(\)\*](#)

#### 57.2.4 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\(\)\*](#)

#### 57.2.5 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\(\)\*](#)

## 57.2.6 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\(\)\*](#)

## 57.2.7 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\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_OnePulse\\_Stop\\_IT\(\)\*](#)

### 57.2.8 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\\_DeInit\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_MspInit\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_MspDeInit\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_Start\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_Stop\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_Encoder\\_Stop\\_DMA\(\)\*](#)

### 57.2.9 IRQ handler management

This section provides Timer IRQ handler function.

This section contains the following APIs:

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

### 57.2.10 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\\_ConfigTI1Input\(\)](#)
- [HAL\\_TIM\\_SlaveConfigSynchronization\(\)](#)
- [HAL\\_TIM\\_SlaveConfigSynchronization\\_IT\(\)](#)
- [HAL\\_TIM\\_ReadCapturedValue\(\)](#)

### 57.2.11 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\(\)](#)

### 57.2.12 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\(\)](#)

### 57.2.13 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>

**57.2.14 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>

**57.2.15 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>

**57.2.16 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>

**57.2.17 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>

**57.2.18 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>

**57.2.19 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	<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>

**57.2.20 HAL\_TIM\_Base\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)</b>
Function Description	Stops the TIM Base generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> pointer to a 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>

**57.2.21 HAL\_TIM\_Base\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>pData:</b> The source Buffer address.</li> <li>• <b>Length:</b> The length of data to be transferred from memory to peripheral.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**57.2.22 HAL\_TIM\_Base\_Stop\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)</b>
Function Description	Stops the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> pointer to a 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>

**57.2.23 HAL\_TIM\_OC\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)</b>
Function Description	Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**57.2.24 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>

**57.2.25 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>

**57.2.26 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>

**57.2.27 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>

**57.2.28 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 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

## Return values

- HAL status

### 57.2.29 HAL\_TIM\_OC\_Start\_IT

## Function Name

**HAL\_StatusTypeDef HAL\_TIM\_OC\_Start\_IT**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)

## Function Description

Starts the TIM Output Compare signal generation in interrupt mode.

## Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channel to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

## Return values

- HAL status

### 57.2.30 HAL\_TIM\_OC\_Stop\_IT

## Function Name

**HAL\_StatusTypeDef HAL\_TIM\_OC\_Stop\_IT**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)

## Function Description

Stops the TIM Output Compare signal generation in interrupt mode.

## Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channel to be disabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

## Return values

- HAL status

### 57.2.31 HAL\_TIM\_OC\_Start\_DMA

## Function Name

**HAL\_StatusTypeDef HAL\_TIM\_OC\_Start\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel, uint32\_t \* pData, uint16\_t Length)

## Function Description

Starts the TIM Output Compare signal generation in DMA mode.

## Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channel to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected
- **pData:** The source Buffer address.

- **Length:** The length of data to be transferred from memory to TIM peripheral
- Return values
- HAL status

### 57.2.32 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

### 57.2.33 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

### 57.2.34 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

### 57.2.35 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

### 57.2.36 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>

### 57.2.37 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>

### 57.2.38 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>

### 57.2.39 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>



**57.2.40 HAL\_TIM\_PWM\_Stop\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> pointer to a 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>

**57.2.41 HAL\_TIM\_PWM\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</b>
Function Description	Starts the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> pointer to a 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>

**57.2.42 HAL\_TIM\_PWM\_Stop\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</b>
Function Description	Stops the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim:</b> pointer to a 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>

**57.2.43 HAL\_TIM\_IC\_Init**

Function Name	<b>HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)</b>
---------------	---

Function Description	Initializes the TIM Input Capture Time base 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>

#### 57.2.44 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 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>

#### 57.2.45 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 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>

#### 57.2.46 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 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>

#### 57.2.47 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 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>

#### 57.2.48 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>

#### 57.2.49 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>

#### 57.2.50 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>

#### 57.2.51 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 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- HAL status

### 57.2.52 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 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

### 57.2.53 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

### 57.2.54 HAL\_TIM\_OnePulse\_DeInit

**Function Name** HAL\_StatusTypeDef HAL\_TIM\_OnePulse\_DeInit (TIM\_HandleTypeDef \* htim)

**Function Description** DeInitializes the TIM One Pulse.

**Parameters**

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- HAL status

### 57.2.55 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>

### 57.2.56 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>

### 57.2.57 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>

### 57.2.58 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>

### 57.2.59 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

### 57.2.60 HAL\_TIM\_OnePulse\_Stop\_IT

**Function Name** **HAL\_StatusTypeDef HAL\_TIM\_OnePulse\_Stop\_IT (TIM\_HandleTypeDef \* htim, uint32\_t OutputChannel)**

**Function Description** Stops the TIM One Pulse signal generation in interrupt mode.

- Parameters**
- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
  - **OutputChannel:** : TIM Channels to be enabled. This parameter can be one of the following values:  
TIM\_CHANNEL\_1: TIM Channel 1  
selectedTIM\_CHANNEL\_2: TIM Channel 2 selected

Return values

- HAL status

### 57.2.61 HAL\_TIM\_Encoder\_Init

**Function Name** **HAL\_StatusTypeDef HAL\_TIM\_Encoder\_Init (TIM\_HandleTypeDef \* htim, TIM\_Encoder\_InitTypeDef \* sConfig)**

**Function Description** Initializes the TIM Encoder Interface and create the associated handle.

- Parameters**
- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
  - **sConfig:** TIM Encoder Interface configuration structure

Return values

- HAL status

### 57.2.62 HAL\_TIM\_Encoder\_DelInit

**Function Name** **HAL\_StatusTypeDef HAL\_TIM\_Encoder\_DelInit (TIM\_HandleTypeDef \* htim)**

**Function Description** DeInitializes the TIM Encoder interface.

- Parameters**
- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- HAL status

### 57.2.63 HAL\_TIM\_Encoder\_MspInit

**Function Name** **void HAL\_TIM\_Encoder\_MspInit (TIM\_HandleTypeDef \* htim)**

**Function Description** Initializes the TIM Encoder Interface MSP.

- Parameters**
- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- None

### 57.2.64 HAL\_TIM\_Encoder\_MspDeInit

Function Name **void HAL\_TIM\_Encoder\_MspDeInit (TIM\_HandleTypeDef \* htim)**

Function Description DeInitializes TIM Encoder Interface MSP.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- None

### 57.2.65 HAL\_TIM\_Encoder\_Start

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Encoder\_Start (TIM\_HandleTypeDef \* htim, uint32\_t Channel)**

Function Description Starts the TIM Encoder Interface.

Parameters

- **htim:** 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

- HAL status

### 57.2.66 HAL\_TIM\_Encoder\_Stop

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Encoder\_Stop (TIM\_HandleTypeDef \* htim, uint32\_t Channel)**

Function Description Stops the TIM Encoder Interface.

Parameters

- **htim:** 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

### 57.2.67 HAL\_TIM\_Encoder\_Start\_IT

Function Name **HAL\_StatusTypeDef HAL\_TIM\_Encoder\_Start\_IT (TIM\_HandleTypeDef \* htim, uint32\_t Channel)**

Function Description Starts the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** 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

- HAL status

### 57.2.68 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

### 57.2.69 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

### 57.2.70 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

- HAL status

### 57.2.71 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

### 57.2.72 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

### 57.2.73 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

### 57.2.74 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>

### 57.2.75 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>

### 57.2.76 HAL\_TIM\_DMABurst\_WriteStart

Function Name	<b>HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (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 memory to 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> <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_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTIM_DMABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_CCERTIM_DMABASE_CNNTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_DMABASE_RCRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_BDTRTIM_DMABASE_DCR</li> </ul>

- **BurstRequestSrc:** 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
  - **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

### 57.2.77 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>

### 57.2.78 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: TIM_DMABASE_CR1TIM_DMABASE_CR2TIM_DMABASE_SMCRTIM_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_BDRTIM_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> </ul>

- **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

### 57.2.79 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

### 57.2.80 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 source  
TIM\_EVENTSOURCE\_CC1: Timer Capture Compare 1 Event source  
TIM\_EVENTSOURCE\_CC2: Timer Capture Compare 2 Event source  
TIM\_EVENTSOURCE\_CC3: Timer Capture Compare 3 Event source  
TIM\_EVENTSOURCE\_CC4: Timer Capture Compare 4 Event source  
TIM\_EVENTSOURCE\_COM: Timer COM event source  
TIM\_EVENTSOURCE\_TRIGGER: Timer Trigger Event source  
TIM\_EVENTSOURCE\_BREAK: Timer Break event source  
TIM\_EVENTSOURCE\_BREAK2: Timer Break2 event source

- Return values**
- HAL status

- Notes**
- TIM6 and TIM7 can only generate an update event.
  - TIM\_EVENTSOURCE\_COM, TIM\_EVENTSOURCE\_BREAK and TIM\_EVENTSOURCE\_BREAK2 are used only with TIM1 and TIM8.

### 57.2.81 HAL\_TIM\_ConfigOCrefClear

**Function Name** HAL\_StatusTypeDef HAL\_TIM\_ConfigOCrefClear (TIM\_HandleTypeDef \* htim, TIM\_ClearInputConfigTypeDef \* sClearInputConfig, uint32\_t Channel)

**Function Description** Configures the OCref clear feature.

Parameters	<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>

### 57.2.82 HAL\_TIM\_ConfigClockSource

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigClockSource</b> (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)
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>

### 57.2.83 HAL\_TIM\_ConfigTI1Input

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigTI1Input</b> (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)
Function Description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	<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 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)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 57.2.84 HAL\_TIM\_SlaveConfigSynchronization

Function Name	<b>HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization</b> (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
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>

### 57.2.85 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>

### 57.2.86 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>

### 57.2.87 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>

### 57.2.88 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>

### 57.2.89 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>

### 57.2.90 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>

### 57.2.91 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>

### 57.2.92 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>

### 57.2.93 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>

### 57.2.94 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>

### 57.2.95 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>

### 57.2.96 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>

### 57.2.97 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>

### 57.2.98 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</li> </ul>



contains the configuration information for TIM module.

Return values

- HAL state

## 57.3 TIM Firmware driver defines

### 57.3.1 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 Clear Input Polarity***

TIM\_CLEARINPUTPOLARITY\_INVERTED      Polarity for ETRx pin

TIM\_CLEARINPUTPOLARITY\_NONINVERTED      Polarity for ETRx pin

#### ***TIM Clear Input Prescaler***

TIM\_CLEARINPUTPRESCALER\_DIV1      No prescaler is used

TIM\_CLEARINPUTPRESCALER\_DIV2      Prescaler for External ETR pin: Capture performed once every 2 events.

TIM\_CLEARINPUTPRESCALER\_DIV4      Prescaler for External ETR pin: Capture performed once every 4 events.

TIM\_CLEARINPUTPRESCALER\_DIV8      Prescaler for External ETR pin: Capture performed once every 8 events.

#### ***TIM Clock Division***

TIM\_CLOCKDIVISION\_DIV1

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 Tlx clock sources

TIM\_CLOCKPOLARITY\_FALLING      Polarity for Tlx clock sources

TIM\_CLOCKPOLARITY\_BOTHEDGE      Polarity for Tlx 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\_EVENTSOURCE\_BREAK2

**TIM Exported Macros**

\_\_HAL\_TIM\_RESET\_HANDLE\_STATE

**Description:**

- Reset TIM handle state.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle

**Return value:**

- None

\_\_HAL\_TIM\_ENABLE

**Description:**

- Enable the TIM peripheral.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle

**Return value:**

- None

---

`__HAL_TIM_URS_ENABLE`**Description:**

- Enable the TIM update source request.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None

`__HAL_TIM_MOE_ENABLE`**Description:**

- Enable the TIM main Output.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None

`TIM_CCER_CCxE_MASK``TIM_CCER_CCxNE_MASK``__HAL_TIM_DISABLE`**Description:**

- Disable the TIM peripheral.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None

`__HAL_TIM_URS_DISABLE`**Description:**

- Disable the TIM update source request.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None

`__HAL_TIM_MOE_DISABLE`**Description:**

- Disable the TIM main Output.

**Parameters:**

- `__HANDLE__`: TIM handle

**Return value:**

- None

`__HAL_TIM_ENABLE_IT``__HAL_TIM_ENABLE_DMA``__HAL_TIM_DISABLE_IT``__HAL_TIM_DISABLE_DMA`

\_\_HAL\_TIM\_GET\_FLAG  
\_\_HAL\_TIM\_CLEAR\_FLAG  
\_\_HAL\_TIM\_GET\_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\_COUNTER

**Description:**

- Sets the TIM Counter Register value on runtime.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.
- \_\_COUNTER\_\_: specifies the Counter register new value.

**Return value:**

- None

\_\_HAL\_TIM\_GET\_COUNTER

**Description:**

- Gets the TIM Counter Register value on runtime.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.

**Return value:**

- None

\_\_HAL\_TIM\_SET\_AUTORELOAD

**Description:**

- Sets the TIM Autoreload Register value on runtime without calling another time any Init function.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.
- \_\_AUTORELOAD\_\_: specifies the Counter register new value.

**Return value:**

- None

\_\_HAL\_TIM\_GET\_AUTORELOAD

**Description:**

- Gets the TIM Autoreload Register value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None

`__HAL_TIM_SET_CLOCKDIVISION`**Description:**

- Sets the TIM Clock Division value on runtime without calling another time any Init function.

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CKD__`: specifies the clock division value. This parameter can be one of the following value:
  - `TIM_CLOCKDIVISION_DIV1`
  - `TIM_CLOCKDIVISION_DIV2`
  - `TIM_CLOCKDIVISION_DIV4`

**Return value:**

- None

`__HAL_TIM_GET_CLOCKDIVISION`**Description:**

- Gets the TIM Clock Division value on runtime.

**Parameters:**

- `__HANDLE__`: TIM handle.

**Return value:**

- None

`__HAL_TIM_SET_ICPRESCALER`**Description:**

- Sets the TIM Input Capture prescaler on runtime without calling another time

**Parameters:**

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: : TIM Channels to be configured. This parameter can be one of the following values:
  - `TIM_CHANNEL_1`: TIM Channel 1 selected
  - `TIM_CHANNEL_2`: TIM Channel 2 selected
  - `TIM_CHANNEL_3`: TIM Channel 3 selected
  - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__ICPSC__`: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:

- TIM\_ICPSC\_DIV1: no prescaler
- TIM\_ICPSC\_DIV2: capture is done once every 2 events
- TIM\_ICPSC\_DIV4: capture is done once every 4 events
- TIM\_ICPSC\_DIV8: capture is done once every 8 events

**Return value:**

- None

**\_\_HAL\_TIM\_GET\_ICPRESCALER****Description:**

- Gets the TIM Input Capture prescaler on runtime.

**Parameters:**

- \_\_HANDLE\_\_: TIM handle.
- \_\_CHANNEL\_\_: : TIM Channels to be configured. This parameter can be one of the following values:
  - TIM\_CHANNEL\_1: get input capture 1 prescaler value
  - TIM\_CHANNEL\_2: get input capture 2 prescaler value
  - TIM\_CHANNEL\_3: get input capture 3 prescaler value
  - TIM\_CHANNEL\_4: get input capture 4 prescaler value

**Return value:**

- None

**\_\_HAL\_TIM\_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\_BREAK2  
 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 Tlx source
TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for Tlx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for Tlx 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\_FAST\_STATE  
IS\_TIM\_OUTPUT\_STATE  
IS\_TIM\_OUTPUTN\_STATE  
IS\_TIM\_OC\_POLARITY  
IS\_TIM\_OCN\_POLARITY  
IS\_TIM\_OCIDLE\_STATE  
IS\_TIM\_OCNIDLE\_STATE  
IS\_TIM\_IC\_POLARITY  
IS\_TIM\_IC\_SELECTION  
IS\_TIM\_IC\_PRESCALER  
IS\_TIM\_OPM\_MODE  
IS\_TIM\_ENCODER\_MODE  
IS\_TIM\_IT  
IS\_TIM\_GET\_IT  
IS\_TIM\_DMA\_SOURCE  
IS\_TIM\_EVENT\_SOURCE  
IS\_TIM\_FLAG  
IS\_TIM\_CLOCKSOURCE  
IS\_TIM\_CLOCKPOLARITY  
IS\_TIM\_CLOCKPRESCALER

IS\_TIM\_CLOCKFILTER  
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\_MSM\_STATE  
IS\_TIM\_TRIGGER\_SELECTION  
IS\_TIM\_INTERNAL\_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 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 Idle State***

TIM\_OCIDLESTATE\_SET

TIM\_OCIDLESTATE\_RESET

***TIM Output Compare N Idle State***

TIM\_OCNIDLESTATE\_SET

TIM\_OCNIDLESTATE\_RESET

***TIM Complementary Output Compare Polarity***

TIM\_OCNPOLARITY\_HIGH

TIM\_OCNPOLARITY\_LOW

***TIM Complementary Output Compare State***

TIM\_OUTPUTNSTATE\_DISABLE

TIM\_OUTPUTNSTATE\_ENABLE

***TIM Output Compare Polarity***

TIM\_OCPOLARITY\_HIGH

TIM\_OCPOLARITY\_LOW

***TIM Output Compare State***

TIM\_OUTPUTSTATE\_DISABLE

TIM\_OUTPUTSTATE\_ENABLE

***TIM Output Fast State***

TIM\_OCFAST\_DISABLE

TIM\_OCFAST\_ENABLE

***TIM 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 TlxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_FALLING	Polarity for TlxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_BOTHEDGE	Polarity for TlxFPx 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

## 58 HAL TIM Extension Driver

### 58.1 TIMEx Firmware driver registers structures

#### 58.1.1 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

#### 58.1.2 TIM\_MasterConfigTypeDef

##### Data Fields

- *uint32\_t MasterOutputTrigger*
- *uint32\_t MasterOutputTrigger2*
- *uint32\_t MasterSlaveMode*

##### Field Documentation

- *uint32\_t TIM\_MasterConfigTypeDef::MasterOutputTrigger*  
Trigger output (TRGO) selection. This parameter can be a value of [TIM\\_Master\\_Mode\\_Selection](#)
- *uint32\_t TIM\_MasterConfigTypeDef::MasterOutputTrigger2*  
Trigger output2 (TRGO2) selection. This parameter can be a value of [TIMEx\\_Master\\_Mode\\_Selection\\_2](#)
- *uint32\_t TIM\_MasterConfigTypeDef::MasterSlaveMode*  
Master/slave mode selection. This parameter can be a value of [TIM\\_Master\\_Slave\\_Mode](#)

### 58.1.3 TIM\_BreakDeadTimeConfigTypeDef

#### Data Fields

- *uint32\_t OffStateRunMode*
- *uint32\_t OffStateIDLEMode*
- *uint32\_t LockLevel*
- *uint32\_t DeadTime*
- *uint32\_t BreakState*
- *uint32\_t BreakPolarity*
- *uint32\_t BreakFilter*
- *uint32\_t Break2State*
- *uint32\_t Break2Polarity*
- *uint32\_t Break2Filter*
- *uint32\_t AutomaticOutput*

#### Field Documentation

- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::OffStateRunMode*  
TIM off state in run mode. This parameter can be a value of [TIM\\_OSSR\\_Off\\_State\\_Selection\\_for\\_Run\\_mode\\_state](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::OffStateIDLEMode*  
TIM off state in IDLE mode. This parameter can be a value of [TIM\\_OSSI\\_Off\\_State\\_Selection\\_for\\_Idle\\_mode\\_state](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::LockLevel*  
TIM Lock level. This parameter can be a value of [TIM\\_Lock\\_level](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::DeadTime*  
TIM dead Time. This parameter can be a number between Min\_Data = 0x00 and Max\_Data = 0xFF
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::BreakState*  
TIM Break State. This parameter can be a value of [TIM\\_Break\\_Input\\_enable\\_disable](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::BreakPolarity*  
TIM Break input polarity. This parameter can be a value of [TIM\\_Break\\_Polarity](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::BreakFilter*  
Specifies the break input filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::Break2State*  
TIM Break2 State This parameter can be a value of [TIMEx\\_Break2\\_Input\\_enable\\_disable](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::Break2Polarity*  
TIM Break2 input polarity This parameter can be a value of [TIMEx\\_Break2\\_Polarity](#)
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::Break2Filter*  
TIM break2 input filter. This parameter can be a number between Min\_Data = 0x0 and Max\_Data = 0xF
- *uint32\_t TIM\_BreakDeadTimeConfigTypeDef::AutomaticOutput*  
TIM Automatic Output Enable state This parameter can be a value of [TIM\\_AOE\\_Bit\\_Set\\_Reset](#)

## 58.2 TIMEx Firmware driver API description

### 58.2.1 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

### 58.2.2 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_OC_N_Start()`, `HAL_TIMEx_OC_N_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()`.



### 58.2.3 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\(\)\*](#)

### 58.2.4 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\\_OC\\_N\\_Start\(\)\*](#)
- [\*HAL\\_TIMEx\\_OC\\_N\\_Stop\(\)\*](#)
- [\*HAL\\_TIMEx\\_OC\\_N\\_Start\\_IT\(\)\*](#)
- [\*HAL\\_TIMEx\\_OC\\_N\\_Stop\\_IT\(\)\*](#)
- [\*HAL\\_TIMEx\\_OC\\_N\\_Start\\_DMA\(\)\*](#)
- [\*HAL\\_TIMEx\\_OC\\_N\\_Stop\\_DMA\(\)\*](#)

### 58.2.5 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\(\)\*](#)

### 58.2.6 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\(\)\*](#)

### 58.2.7 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\\_ConfigCommutationEvent\(\)\*](#)
- [\*HAL\\_TIMEx\\_ConfigCommutationEvent\\_IT\(\)\*](#)
- [\*HAL\\_TIMEx\\_ConfigCommutationEvent\\_DMA\(\)\*](#)
- [\*HAL\\_TIM\\_OC\\_ConfigChannel\(\)\*](#)
- [\*HAL\\_TIM\\_PWM\\_ConfigChannel\(\)\*](#)
- [\*HAL\\_TIM\\_ConfigOCrefClear\(\)\*](#)
- [\*HAL\\_TIMEx\\_MasterConfigSynchronization\(\)\*](#)
- [\*HAL\\_TIMEx\\_ConfigBreakDeadTime\(\)\*](#)
- [\*HAL\\_TIMEx\\_RemapConfig\(\)\*](#)
- [\*HAL\\_TIMEx\\_GroupChannel5\(\)\*](#)

## 58.2.8 Extension Callbacks functions

This section provides Extension TIM callback functions:

- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:

- [HAL\\_TIMEx\\_CommutationCallback\(\)](#)
- [HAL\\_TIMEx\\_BreakCallback\(\)](#)
- [HAL\\_TIMEx\\_DMACommutationCplt\(\)](#)

## 58.2.9 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\(\)](#)

### 58.2.10 HAL\_TIMEx\_HallSensor\_Init

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init</b> (TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)
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>

### 58.2.11 HAL\_TIMEx\_HallSensor\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit</b> (TIM_HandleTypeDef * htim)
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>

### 58.2.12 HAL\_TIMEx\_HallSensor\_MspInit

Function Name	<b>void HAL_TIMEx_HallSensor_MspInit</b> (TIM_HandleTypeDef * htim)
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>

### 58.2.13 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>

#### 58.2.14 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>

#### 58.2.15 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>

#### 58.2.16 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>

#### 58.2.17 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>

#### 58.2.18 HAL\_TIMEx\_HallSensor\_Start\_DMA

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

---

(TIM\_HandleTypeDef \* htim, uint32\_t \* pData, uint16\_t Length)

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>

### 58.2.19 HAL\_TIMEx\_HallSensor\_Stop\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA</b> (TIM_HandleTypeDef * htim)
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>

### 58.2.20 HAL\_TIMEx\_OCN\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Start</b> (TIM_HandleTypeDef * htim, uint32_t Channel)
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>

### 58.2.21 HAL\_TIMEx\_OCN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop</b> (TIM_HandleTypeDef * htim, uint32_t Channel)
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>

**58.2.22 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>

**58.2.23 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> <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>

**58.2.24 HAL\_TIMEx\_OCN\_Start\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_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 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>

**58.2.25 HAL\_TIMEx\_OCN\_Stop\_DMA**

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA</b> (TIM_HandleTypeDef * htim, uint32_t Channel)
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>

### 58.2.26 HAL\_TIMEx\_PWMN\_Start

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start</b> (TIM_HandleTypeDef * htim, uint32_t Channel)
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>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 58.2.27 HAL\_TIMEx\_PWMN\_Stop

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop</b> (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>Channel</b>: TIM Channel to be disabled. This parameter can 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>

### 58.2.28 HAL\_TIMEx\_PWMN\_Start\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT</b> (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: pointer to a 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

### 58.2.29 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

### 58.2.30 HAL\_TIMEx\_PWMN\_Start\_DMA

Function Name

**HAL\_StatusTypeDef HAL\_TIMEx\_PWMN\_Start\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel, uint32\_t \* pData, uint16\_t Length)

Function Description

Starts the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channel to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- HAL status

### 58.2.31 HAL\_TIMEx\_PWMN\_Stop\_DMA

Function Name

**HAL\_StatusTypeDef HAL\_TIMEx\_PWMN\_Stop\_DMA**  
(TIM\_HandleTypeDef \* htim, uint32\_t Channel)

Function Description

Stops the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Channel:** TIM Channel to be disabled. This parameter can



be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected  
TIM\_CHANNEL\_3: TIM Channel 3 selected  
TIM\_CHANNEL\_4: TIM Channel 4 selected

## Return values

- HAL status

### 58.2.32 HAL\_TIMEx\_OnePulseN\_Start

## Function Name

**HAL\_StatusTypeDef HAL\_TIMEx\_OnePulseN\_Start**  
(TIM\_HandleTypeDef \* htim, uint32\_t OutputChannel)

## Function Description

Starts the TIM One Pulse signal generation on the complementary output.

## Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **OutputChannel:** TIM Channel to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected

## Return values

- HAL status

### 58.2.33 HAL\_TIMEx\_OnePulseN\_Stop

## Function Name

**HAL\_StatusTypeDef HAL\_TIMEx\_OnePulseN\_Stop**  
(TIM\_HandleTypeDef \* htim, uint32\_t OutputChannel)

## Function Description

Stops the TIM One Pulse signal generation on the complementary output.

## Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **OutputChannel:** TIM Channel to be disabled. This parameter can be one of the following values:  
TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected

## Return values

- HAL status

### 58.2.34 HAL\_TIMEx\_OnePulseN\_Start\_IT

## Function Name

**HAL\_StatusTypeDef HAL\_TIMEx\_OnePulseN\_Start\_IT**  
(TIM\_HandleTypeDef \* htim, uint32\_t OutputChannel)

## Function Description

Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.

## Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **OutputChannel:** TIM Channel to be enabled. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selected  
TIM\_CHANNEL\_2: TIM Channel 2 selected

## Return values

- HAL status

### 58.2.35 HAL\_TIMEx\_OnePulseN\_Stop\_IT



Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT</b> (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
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>

### 58.2.36 HAL\_TIMEx\_ConfigCommutationEvent

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent</b> (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
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 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> </ul>

### 58.2.37 HAL\_TIMEx\_ConfigCommutationEvent\_IT

Function Name	<b>HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT</b> (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function Description	Configure the TIM commutation event sequence with interrupt.

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> </ul>

### 58.2.38 HAL\_TIMEx\_ConfigCommutationEvent\_DMA

Function Name	<b>HAL_StatusTypeDef</b> <b>HAL_TIMEx_ConfigCommutationEvent_DMA</b> <b>(TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)</b>
Function Description	Configure the TIM commutation event sequence with DMA.
Parameters	<ul style="list-style-type: none"> <li>• <b>htim</b>: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.</li> <li>• <b>InputTrigger</b>: the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values: TIM_TS_ITR0: 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)</li> </ul>

configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

- : The user should configure the DMA in his own software, in This function only the COMDE bit is set

### 58.2.39 HAL\_TIM\_OC\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel</b> (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	<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 Output Compare 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>

### 58.2.40 HAL\_TIM\_PWM\_ConfigChannel

Function Name	<b>HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel</b> (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>

### 58.2.41 HAL\_TIM\_ConfigOCrefClear

Function Name	<b>HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear</b> (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)
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</li> </ul>

TIM\_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.

- **Channel:** specifies the TIM Channel. This parameter can be one of the following values: TIM\_CHANNEL\_1: TIM Channel 1 selectedTIM\_CHANNEL\_2: TIM Channel 2 selectedTIM\_CHANNEL\_3: TIM Channel 3 selectedTIM\_CHANNEL\_4: TIM Channel 4 selected

Return values

- HAL status

## 58.2.42 HAL\_TIMEx\_MasterConfigSynchronization

**Function Name** HAL\_StatusTypeDef  
HAL\_TIMEx\_MasterConfigSynchronization  
(TIM\_HandleTypeDef \* htim, TIM\_MasterConfigTypeDef \* sMasterConfig)

**Function Description** Configures the TIM in master mode.

**Parameters**

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **sMasterConfig:** pointer to a TIM\_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values

- HAL status

## 58.2.43 HAL\_TIMEx\_ConfigBreakDeadTime

**Function Name** HAL\_StatusTypeDef HAL\_TIMEx\_ConfigBreakDeadTime  
(TIM\_HandleTypeDef \* htim,  
TIM\_BreakDeadTimeConfigTypeDef \* sBreakDeadTimeConfig)

**Function Description** Configures the Break feature, dead time, Lock level, OSSR/OSSI State and the AOE(automatic output enable).

**Parameters**

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **sBreakDeadTimeConfig:** pointer to a TIM\_ConfigBreakDeadConfig\_TypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values

- HAL status

## 58.2.44 HAL\_TIMEx\_RemapConfig

**Function Name** HAL\_StatusTypeDef HAL\_TIMEx\_RemapConfig  
(TIM\_HandleTypeDef \* htim, uint32\_t Remap)

**Function Description** Configures the TIM2, TIM5 and TIM11 Remapping input capabilities.

**Parameters**

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.
- **Remap:** specifies the TIM input remapping source. This parameter can be one of the following values:  
TIM\_TIM2\_TIM8\_TRGO: TIM2 ITR1 input is connected to TIM8 Trigger output(default)  
TIM\_TIM2\_ETH\_PTP: TIM2 ITR1

input is connected to ETH PTP trigger output. TIM\_TIM2\_USBFS\_SOF: TIM2 ITR1 input is connected to USB FS SOF. TIM\_TIM2\_USBHS\_SOF: TIM2 ITR1 input is connected to USB HS SOF. TIM\_TIM5\_GPIO: TIM5 CH4 input is connected to dedicated Timer pin(default) TIM\_TIM5\_LSI: TIM5 CH4 input is connected to LSI clock. TIM\_TIM5\_LSE: TIM5 CH4 input is connected to LSE clock. TIM\_TIM5\_RTC: TIM5 CH4 input is connected to RTC Output event. TIM\_TIM11\_GPIO: TIM11 CH4 input is connected to dedicated Timer pin(default) TIM\_TIM11\_SPDIF: SPDIF Frame synchronous. TIM\_TIM11\_HSE: TIM11 CH4 input is connected to HSE\_RTC clock (HSE divided by a programmable prescaler) TIM\_TIM11\_MCO1: TIM11 CH1 input is connected to MCO1

Return values

- HAL status

### 58.2.45 HAL\_TIMEx\_GroupChannel5

Function Name **HAL\_StatusTypeDef HAL\_TIMEx\_GroupChannel5 (TIM\_HandleTypeDef \* htim, uint32\_t OCRef)**

Function Description Group channel 5 and channel 1, 2 or 3.

Parameters

- **htim:** TIM handle.
- **OCRef:** specifies the reference signal(s) the OC5REF is combined with. This parameter can be any combination of the following values: TIM\_GROUPCH5\_NONE: No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC  
TIM\_GROUPCH5\_OC1REFC: OC1REFC is the logical AND of OC1REFC and OC5REF  
TIM\_GROUPCH5\_OC2REFC: OC2REFC is the logical AND of OC2REFC and OC5REF  
TIM\_GROUPCH5\_OC3REFC: OC3REFC is the logical AND of OC3REFC and OC5REF

Return values

- HAL status

### 58.2.46 HAL\_TIMEx\_CommutationCallback

Function Name **void HAL\_TIMEx\_CommutationCallback (TIM\_HandleTypeDef \* htim)**

Function Description Hall commutation changed callback in non blocking mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- None

### 58.2.47 HAL\_TIMEx\_BreakCallback

Function Name **void HAL\_TIMEx\_BreakCallback (TIM\_HandleTypeDef \* htim)**

Function Description Hall Break detection callback in non blocking mode.

Parameters

- **htim:** pointer to a TIM\_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- None

**58.2.48 HAL\_TIMEx\_DMACommutationCplt**

Function Name	<b>void HAL_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>

**58.2.49 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>

**58.3 TIMEx Firmware driver defines****58.3.1 TIMEx*****TIMEx Break input 2 Enable***

TIM\_BREAK2\_DISABLE

TIM\_BREAK2\_ENABLE

***TIMEx Break2 Polarity***

TIM\_BREAK2POLARITY\_LOW

TIM\_BREAK2POLARITY\_HIGH

***TIMEx Channel***

TIM\_CHANNEL\_1

TIM\_CHANNEL\_2

TIM\_CHANNEL\_3

TIM\_CHANNEL\_4

TIM\_CHANNEL\_5

TIM\_CHANNEL\_6

TIM\_CHANNEL\_ALL

***TIMEx Clear Input Source***

TIM\_CLEARINPUTSOURCE\_ETR

TIM\_CLEARINPUTSOURCE\_OCREFCLR

TIM\_CLEARINPUTSOURCE\_NONE

***TIMEx Exported Macros***

**\_\_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
  - **TIM\_CHANNEL\_5**: TIM Channel 5 selected
  - **TIM\_CHANNEL\_6**: TIM Channel 6 selected
- **\_\_COMPARE\_\_**: specifies the Capture Compare register new value.

**Return value:**

- None

**\_\_HAL\_TIM\_GET\_COMPARE****Description:**

- Gets the TIM Capture Compare Register value on runtime.

**Parameters:**

- **\_\_HANDLE\_\_**: TIM handle.
- **\_\_CHANNEL\_\_**: : TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - **TIM\_CHANNEL\_1**: get capture/compare 1 register value
  - **TIM\_CHANNEL\_2**: get capture/compare 2 register value
  - **TIM\_CHANNEL\_3**: get capture/compare 3 register value
  - **TIM\_CHANNEL\_4**: get capture/compare 4 register value
  - **TIM\_CHANNEL\_5**: get capture/compare 5 register value
  - **TIM\_CHANNEL\_6**: get capture/compare 6 register value

**Return value:**

- None

***TIMEx Group Channel 5 and Channel 1, 2 or 3***

TIM\_GROUPCH5\_NONE

TIM\_GROUPCH5\_OC1REFC

TIM\_GROUPCH5\_OC2REFC

TIM\_GROUPCH5\_OC3REFC



***TIMEx Master Mode Selection 2 (TRGO2)***

TIM\_TRGO2\_RESET  
TIM\_TRGO2\_ENABLE  
TIM\_TRGO2\_UPDATE  
TIM\_TRGO2\_OC1  
TIM\_TRGO2\_OC1REF  
TIM\_TRGO2\_OC2REF  
TIM\_TRGO2\_OC3REF  
TIM\_TRGO2\_OC4REF  
TIM\_TRGO2\_OC5REF  
TIM\_TRGO2\_OC6REF  
TIM\_TRGO2\_OC4REF\_RISINGFALLING  
TIM\_TRGO2\_OC6REF\_RISINGFALLING  
TIM\_TRGO2\_OC4REF\_RISING\_OC6REF\_RISING  
TIM\_TRGO2\_OC4REF\_RISING\_OC6REF\_FALLING  
TIM\_TRGO2\_OC5REF\_RISING\_OC6REF\_RISING  
TIM\_TRGO2\_OC5REF\_RISING\_OC6REF\_FALLING

***TIMEx Output Compare and PWM Modes***

TIM\_OCMODE\_TIMING  
TIM\_OCMODE\_ACTIVE  
TIM\_OCMODE\_INACTIVE  
TIM\_OCMODE\_TOGGLE  
TIM\_OCMODE\_PWM1  
TIM\_OCMODE\_PWM2  
TIM\_OCMODE\_FORCED\_ACTIVE  
TIM\_OCMODE\_FORCED\_INACTIVE  
TIM\_OCMODE\_RETRIGERRABLE\_OPM1  
TIM\_OCMODE\_RETRIGERRABLE\_OPM2  
TIM\_OCMODE\_COMBINED\_PWM1  
TIM\_OCMODE\_COMBINED\_PWM2  
TIM\_OCMODE\_ASSYMETRIC\_PWM1  
TIM\_OCMODE\_ASSYMETRIC\_PWM2

***TIMEx Private Macros***

IS\_TIM\_CHANNELS  
IS\_TIM\_PWMI\_CHANNELS  
IS\_TIM\_OPM\_CHANNELS

IS\_TIM\_COMPLEMENTARY\_CHANNELS

IS\_TIM\_PWM\_MODE

IS\_TIM\_OC\_MODE

IS\_TIM\_REMAP

IS\_TIM\_DEADTIME

IS\_TIM\_BREAK\_FILTER

IS\_TIM\_CLEARINPUT\_SOURCE

IS\_TIM\_BREAK2\_STATE

IS\_TIM\_BREAK2\_POLARITY

IS\_TIM\_GROUPCH5

IS\_TIM\_TRGO2\_SOURCE

IS\_TIM\_SLAVE\_MODE

***TIMEx 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

TIM\_TIM11\_SPDIFRX

TIM\_TIM11\_HSE

TIM\_TIM11\_MCO1

***TIMEx Slave mode***

TIM\_SLAVEMODE\_DISABLE

TIM\_SLAVEMODE\_RESET

TIM\_SLAVEMODE\_GATED

TIM\_SLAVEMODE\_TRIGGER

TIM\_SLAVEMODE\_EXTERNAL1

TIM\_SLAVEMODE\_COMBINED\_RESETTRIGGER

## 59 HAL UART Generic Driver

### 59.1 UART Firmware driver registers structures

#### 59.1.1 UART\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t HwFlowCtl*
- *uint32\_t OverSampling*
- *uint32\_t OneBitSampling*

##### Field Documentation

- ***uint32\_t UART\_InitTypeDef::BaudRate***  
This member configures the UART communication baud rate. The baud rate register is computed using the following formula: If oversampling is 16 or in LIN mode, Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate))) If oversampling is 8, Baud Rate Register[15:4] = ((2 \* PCLKx) / ((huart->Init.BaudRate))) [15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 \* PCLKx) / ((huart->Init.BaudRate)))) [3:0] >> 1
- ***uint32\_t UART\_InitTypeDef::WordLength***  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [UARTEx\\_Word\\_Length](#)
- ***uint32\_t UART\_InitTypeDef::StopBits***  
Specifies the number of stop bits transmitted. This parameter can be a value of [UART\\_Stop\\_Bits](#)
- ***uint32\_t UART\_InitTypeDef::Parity***  
Specifies the parity mode. This parameter can be a value of [UART\\_Parity](#)  
**Note:** When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32\_t UART\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [UART\\_Mode](#)
- ***uint32\_t UART\_InitTypeDef::HwFlowCtl***  
Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of [UART\\_Hardware\\_Flow\\_Control](#)
- ***uint32\_t UART\_InitTypeDef::OverSampling***  
Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of [UART\\_Over\\_Sampling](#)
- ***uint32\_t UART\_InitTypeDef::OneBitSampling***  
Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of [UART\\_OneBit\\_Sampling](#)

### 59.1.2 UART\_AdvFeatureInitTypeDef

#### Data Fields

- *uint32\_t AdvFeatureInit*
- *uint32\_t TxPinLevelInvert*
- *uint32\_t RxPinLevelInvert*
- *uint32\_t DataInvert*
- *uint32\_t Swap*
- *uint32\_t OverrunDisable*
- *uint32\_t DMADisableonRxError*
- *uint32\_t AutoBaudRateEnable*
- *uint32\_t AutoBaudRateMode*
- *uint32\_t MSBFirst*

#### Field Documentation

- *uint32\_t UART\_AdvFeatureInitTypeDef::AdvFeatureInit*  
Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of [UART\\_Advanced\\_Features\\_Initialization\\_Type](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::TxPinLevelInvert*  
Specifies whether the TX pin active level is inverted. This parameter can be a value of [UART\\_Tx\\_Inv](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::RxPinLevelInvert*  
Specifies whether the RX pin active level is inverted. This parameter can be a value of [UART\\_Rx\\_Inv](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::DataInvert*  
Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of [UART\\_Data\\_Inv](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::Swap*  
Specifies whether TX and RX pins are swapped. This parameter can be a value of [UART\\_Rx\\_Tx\\_Swap](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::OverrunDisable*  
Specifies whether the reception overrun detection is disabled. This parameter can be a value of [UART\\_Overrun\\_Disable](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::DMADisableonRxError*  
Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of [UART\\_DMA\\_Disable\\_on\\_Rx\\_Error](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::AutoBaudRateEnable*  
Specifies whether auto Baud rate detection is enabled. This parameter can be a value of [UART\\_AutoBaudRate\\_Enable](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::AutoBaudRateMode*  
If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of [UART\\_AutoBaud\\_Rate\\_Mode](#)
- *uint32\_t UART\_AdvFeatureInitTypeDef::MSBFirst*  
Specifies whether MSB is sent first on UART line. This parameter can be a value of [UART\\_MSB\\_First](#)

### 59.1.3 UART\_HandleTypeDef

**Data Fields**

- **USART\_TypeDef \* Instance**
- **UART\_InitTypeDef Init**
- **UART\_AdvFeatureInitTypeDef AdvancedInit**
- **uint8\_t \* pTxBuffPtr**
- **uint16\_t TxXferSize**
- **uint16\_t TxXferCount**
- **uint8\_t \* pRxBuffPtr**
- **uint16\_t RxXferSize**
- **uint16\_t RxXferCount**
- **uint16\_t Mask**
- **DMA\_HandleTypeDef \* hdmatx**
- **DMA\_HandleTypeDef \* hdmarx**
- **HAL\_LockTypeDef Lock**
- **\_\_IO HAL\_UART\_StateTypeDef State**
- **\_\_IO uint32\_t ErrorCode**

**Field Documentation**

- **USART\_TypeDef\* UART\_HandleTypeDef::Instance**  
UART registers base address
- **UART\_InitTypeDef UART\_HandleTypeDef::Init**  
UART communication parameters
- **UART\_AdvFeatureInitTypeDef UART\_HandleTypeDef::AdvancedInit**  
UART Advanced Features initialization parameters
- **uint8\_t\* UART\_HandleTypeDef::pTxBuffPtr**  
Pointer to UART Tx transfer Buffer
- **uint16\_t UART\_HandleTypeDef::TxXferSize**  
UART Tx Transfer size
- **uint16\_t UART\_HandleTypeDef::TxXferCount**  
UART Tx Transfer Counter
- **uint8\_t\* UART\_HandleTypeDef::pRxBuffPtr**  
Pointer to UART Rx transfer Buffer
- **uint16\_t UART\_HandleTypeDef::RxXferSize**  
UART Rx Transfer size
- **uint16\_t UART\_HandleTypeDef::RxXferCount**  
UART Rx Transfer Counter
- **uint16\_t UART\_HandleTypeDef::Mask**  
UART Rx RDR register mask
- **DMA\_HandleTypeDef\* UART\_HandleTypeDef::hdmatx**  
UART Tx DMA Handle parameters
- **DMA\_HandleTypeDef\* UART\_HandleTypeDef::hdmarx**  
UART Rx DMA Handle parameters
- **HAL\_LockTypeDef UART\_HandleTypeDef::Lock**  
Locking object
- **\_\_IO HAL\_UART\_StateTypeDef UART\_HandleTypeDef::State**  
UART communication state
- **\_\_IO uint32\_t UART\_HandleTypeDef::ErrorCode**  
UART Error code

## 59.2 UART Firmware driver API description

### 59.2.1 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\_IT : Clears the specified UART ISR 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

### 59.2.2 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\\_DeInit\(\)\*](#)
- [\*HAL\\_UART\\_MspInit\(\)\*](#)
- [\*HAL\\_UART\\_MspDeInit\(\)\*](#)

### 59.2.3 IO operation functions

This section contains the following APIs:

- [\*HAL\\_UART\\_Transmit\(\)\*](#)
- [\*HAL\\_UART\\_Receive\(\)\*](#)
- [\*HAL\\_UART\\_Transmit\\_IT\(\)\*](#)
- [\*HAL\\_UART\\_Receive\\_IT\(\)\*](#)
- [\*HAL\\_UART\\_Transmit\\_DMA\(\)\*](#)
- [\*HAL\\_UART\\_Receive\\_DMA\(\)\*](#)
- [\*HAL\\_UART\\_DMABasePause\(\)\*](#)
- [\*HAL\\_UART\\_DMABaseResume\(\)\*](#)
- [\*HAL\\_UART\\_DMABaseStop\(\)\*](#)
- [\*HAL\\_UART\\_IRQHandler\(\)\*](#)
- [\*UART\\_WaitOnFlagUntilTimeout\(\)\*](#)
- [\*HAL\\_UART\\_TxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_TxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_RxCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_RxHalfCpltCallback\(\)\*](#)
- [\*HAL\\_UART\\_ErrorCallback\(\)\*](#)

### 59.2.4 Peripheral Control functions

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



- HAL\_UART\_GetState() API is helpful to check in run-time the state of the UART peripheral.
- HAL\_MultiProcessor\_EnableMuteMode() API enables mute mode
- HAL\_MultiProcessor\_DisableMuteMode() API disables mute mode
- HAL\_MultiProcessor\_EnterMuteMode() API enters mute mode
- HAL\_MultiProcessor\_EnableMuteMode() API enables mute mode
- UART\_SetConfig() API configures the UART peripheral
- UART\_AdvFeatureConfig() API optionally configures the UART advanced features
- UART\_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- HAL\_HalfDuplex\_EnableTransmitter() API disables receiver and enables transmitter
- HAL\_HalfDuplex\_EnableReceiver() API disables transmitter and enables receiver
- HAL\_LIN\_SendBreak() API transmits the break characters

This section contains the following APIs:

- [\*HAL\\_MultiProcessor\\_EnableMuteMode\(\)\*](#)
- [\*HAL\\_MultiProcessor\\_DisableMuteMode\(\)\*](#)
- [\*HAL\\_MultiProcessor\\_EnterMuteMode\(\)\*](#)
- [\*HAL\\_UART\\_GetState\(\)\*](#)
- [\*HAL\\_UART\\_GetError\(\)\*](#)
- [\*UART\\_SetConfig\(\)\*](#)
- [\*UART\\_AdvFeatureConfig\(\)\*](#)
- [\*UART\\_CheckIdleState\(\)\*](#)
- [\*HAL\\_HalfDuplex\\_EnableTransmitter\(\)\*](#)
- [\*HAL\\_HalfDuplex\\_EnableReceiver\(\)\*](#)
- [\*HAL\\_LIN\\_SendBreak\(\)\*](#)

### 59.2.5 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 UART_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 59.2.6 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 UART_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 59.2.7 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 <code>UART_InitTypeDef</code> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> <li>• <b>BreakDetectLength:</b> specifies the LIN break detection length. This parameter can be one of the following values:  <code>UART_LINBREAKDETECTLENGTH_10B</code>: 10-bit break detection  <code>UART_LINBREAKDETECTLENGTH_11B</code>: 11-bit break detection</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 59.2.8 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 multiprocessor mode according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> <li>• <b>Address:</b> UART node address (4-, 6-, 7- or 8-bit long)</li> <li>• <b>WakeUpMethod:</b> specifies the UART wakeup method. This parameter can be one of the following values:  <code>UART_WAKEUPMETHOD_IDLELINE</code>: WakeUp by an idle line detection  <code>UART_WAKEUPMETHOD_ADDRESSMARK</code>: WakeUp by an address mark</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>
Notes	<ul style="list-style-type: none"> <li>• If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function.</li> <li>• If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection</li> </ul>

### 59.2.9 HAL\_UART\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)</b>
Function Description	DeInitializes the UART peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 59.2.10 HAL\_UART\_MspInit

Function Name	<b>void HAL_UART_MspInit (UART_HandleTypeDef * huart)</b>
Function Description	UART MSP Init.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**59.2.11 HAL\_UART\_MspDeInit**

Function Name	<b>void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)</b>
Function Description	UART MSP DeInit.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

**59.2.12 HAL\_UART\_Transmit**

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, 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>huart:</b> uart handle</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>

**59.2.13 HAL\_UART\_Receive**

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, 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>huart:</b> uart handle</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>

**59.2.14 HAL\_UART\_Transmit\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</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>

**59.2.15 HAL\_UART\_Receive\_IT**

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</b>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> uart handle</li> </ul>

- **pData:** pointer to data buffer
  - **Size:** amount of data to be received
- Return values
- HAL status

### 59.2.16 HAL\_UART\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_UART_Transmit_DMA</b> (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> uart handle</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>

### 59.2.17 HAL\_UART\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_UART_Receive_DMA</b> (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> uart handle</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 UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li></ul>

### 59.2.18 HAL\_UART\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAPause</b> (UART_HandleTypeDef * huart)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 59.2.19 HAL\_UART\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAResume</b> (UART_HandleTypeDef * huart)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>huart:</b> UART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 59.2.20 HAL\_UART\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_UART_DMAStop</b> (UART_HandleTypeDef * huart)
---------------	---

Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 59.2.21 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> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 59.2.22 UART\_WaitOnFlagUntilTimeout

Function Name	<b>HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Timeout)</b>
Function Description	This function handles UART Communication Timeout.
Parameters	<ul style="list-style-type: none"> <li>• <b>huart:</b> UART handle</li> <li>• <b>Flag:</b> specifies the UART flag to check.</li> <li>• <b>Status:</b> The new Flag status (SET or RESET).</li> <li>• <b>Timeout:</b> Timeout duration</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 59.2.23 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> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 59.2.24 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> UART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• None</li> </ul>

### 59.2.25 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> uart handle</li> </ul>

Return values

- None

### 59.2.26 HAL\_UART\_RxHalfCpltCallback

Function Name **void HAL\_UART\_RxHalfCpltCallback (UART\_HandleTypeDef \* huart)**

Function Description Rx Half Transfer completed callbacks.

Parameters

- **huart:** UART handle

Return values

- None

### 59.2.27 HAL\_UART\_ErrorCallback

Function Name **void HAL\_UART\_ErrorCallback (UART\_HandleTypeDef \* huart)**

Function Description UART error callbacks.

Parameters

- **huart:** uart handle

Return values

- None

### 59.2.28 HAL\_MultiProcessor\_EnableMuteMode

Function Name **HAL\_StatusTypeDef HAL\_MultiProcessor\_EnableMuteMode (UART\_HandleTypeDef \* huart)**

Function Description Enable UART in mute mode (doesn't mean UART enters mute mode; to enter mute mode, HAL\_MultiProcessor\_EnterMuteMode() API must be called)

Parameters

- **huart:** UART handle

Return values

- HAL status

### 59.2.29 HAL\_MultiProcessor\_DisableMuteMode

Function Name **HAL\_StatusTypeDef HAL\_MultiProcessor\_DisableMuteMode (UART\_HandleTypeDef \* huart)**

Function Description Disable UART mute mode (doesn't mean it actually wakes up the software, as it may not have been in mute mode at this very moment).

Parameters

- **huart:** uart handle

Return values

- HAL status

### 59.2.30 HAL\_MultiProcessor\_EnterMuteMode

Function Name **void HAL\_MultiProcessor\_EnterMuteMode (UART\_HandleTypeDef \* huart)**

Function Description Enter UART mute mode (means UART actually enters mute mode).

Parameters

- **huart:** uart handle

Return values

- HAL status

**59.2.31 HAL\_UART\_GetState**

Function Name	<b>HAL_StatusTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)</b>
Function Description	return the UART state
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL state</li> </ul>

**59.2.32 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>

**59.2.33 UART\_SetConfig**

Function Name	<b>HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)</b>
Function Description	Configure the UART peripheral.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**59.2.34 UART\_AdvFeatureConfig**

Function Name	<b>void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)</b>
Function Description	Configure the UART peripheral advanced features.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>

**59.2.35 UART\_CheckIdleState**

Function Name	<b>HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)</b>
Function Description	Check the UART Idle State.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> uart handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>HAL status</li> </ul>

**59.2.36 HAL\_HalfDuplex\_EnableTransmitter**

Function Name	<b>HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)</b>
Function Description	Enables the UART transmitter and disables the UART receiver.
Parameters	<ul style="list-style-type: none"> <li><b>huart:</b> UART handle</li> </ul>

- Return values
- HAL status
  - None

### 59.2.37 HAL\_HalfDuplex\_EnableReceiver

- Function Name **HAL\_StatusTypeDef HAL\_HalfDuplex\_EnableReceiver (UART\_HandleTypeDef \* huart)**
- Function Description Enables the UART receiver and disables the UART transmitter.
- Parameters
- **huart:** UART handle
- Return values
- HAL status

### 59.2.38 HAL\_LIN\_SendBreak

- Function Name **HAL\_StatusTypeDef HAL\_LIN\_SendBreak (UART\_HandleTypeDef \* huart)**
- Function Description Transmits break characters.
- Parameters
- **huart:** UART handle
- Return values
- HAL status

### 59.2.39 HAL\_UART\_GetState

- Function Name **HAL\_UART\_StateTypeDef HAL\_UART\_GetState (UART\_HandleTypeDef \* huart)**
- Function Description return the UART state
- Parameters
- **huart:** uart handle
- Return values
- HAL state

### 59.2.40 HAL\_UART\_GetError

- Function Name **uint32\_t HAL\_UART\_GetError (UART\_HandleTypeDef \* huart)**
- Function Description Return the UART error code.
- Parameters
- **huart:** : pointer to a UART\_HandleTypeDef structure that contains the configuration information for the specified UART.
- Return values
- UART Error Code

## 59.3 UART Firmware driver defines

### 59.3.1 UART

#### *UART Advanced Feature Initialization Type*

UART\_ADVFEATURE\_NO\_INIT  
 UART\_ADVFEATURE\_TXINVERT\_INIT  
 UART\_ADVFEATURE\_RXINVERT\_INIT  
 UART\_ADVFEATURE\_DATAINVERT\_INIT  
 UART\_ADVFEATURE\_SWAP\_INIT



UART\_ADVFEATURE\_RXOVERRUNDISABLE\_INIT

UART\_ADVFEATURE\_DMADISABLEONERROR\_INIT

UART\_ADVFEATURE\_AUTOBAUDRATE\_INIT

UART\_ADVFEATURE\_MSBFIRST\_INIT

**UART Advanced Feature Auto BaudRate Enable**

UART\_ADVFEATURE\_AUTOBAUDRATE\_DISABLE

UART\_ADVFEATURE\_AUTOBAUDRATE\_ENABLE

**UART Advanced Feature AutoBaud Rate Mode**

UART\_ADVFEATURE\_AUTOBAUDRATE\_ONSTARTBIT

UART\_ADVFEATURE\_AUTOBAUDRATE\_ONFALLINGEDGE

UART\_ADVFEATURE\_AUTOBAUDRATE\_ON0X7FFRAME

UART\_ADVFEATURE\_AUTOBAUDRATE\_ON0X55FRAME

**UART Driver Enable Assertion Time LSB Position In CR1 Register**

UART\_CR1\_DEAT\_ADDRESS\_LSB\_POS

**UART Driver Enable DeAssertion Time LSB Position In CR1 Register**

UART\_CR1\_DEDT\_ADDRESS\_LSB\_POS

**UART Address-matching LSB Position In CR2 Register**

UART\_CR2\_ADDRESS\_LSB\_POS

**UART Advanced Feature Binary Data Inversion**

UART\_ADVFEATURE\_DATAINV\_DISABLE

UART\_ADVFEATURE\_DATAINV\_ENABLE

**UART Advanced Feature DMA Disable On Rx Error**

UART\_ADVFEATURE\_DMA\_ENABLEONRXERROR

UART\_ADVFEATURE\_DMA\_DISABLEONRXERROR

**UART DMA Rx**

UART\_DMA\_RX\_DISABLE

UART\_DMA\_RX\_ENABLE

**UART DMA Tx**

UART\_DMA\_TX\_DISABLE

UART\_DMA\_TX\_ENABLE

**UART DriverEnable Polarity**

UART\_DE\_POLARITY\_HIGH

UART\_DE\_POLARITY\_LOW

**UART Error Definition**

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\_\_**: UART handle.

**Return value:**

- None

**\_\_HAL\_UART\_FLUSH\_DRREGISTER**

**Description:**

- Flush the UART Data registers.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle.

**\_\_HAL\_UART\_CLEAR\_IT**

**Description:**

- Clears the specified UART ISR flag, in setting the proper ICR register flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the UART Handle.
- **\_\_FLAG\_\_**: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - UART\_CLEAR\_PEF: Parity Error Clear Flag
  - UART\_CLEAR\_FEF: Framing Error Clear Flag
  - UART\_CLEAR\_NEF: Noise detected Clear Flag
  - UART\_CLEAR\_OREF: OverRun Error Clear Flag
  - UART\_CLEAR\_IDLEF: IDLE line detected Clear Flag
  - UART\_CLEAR\_TCF: Transmission Complete Clear Flag
  - UART\_CLEAR\_LBDF: LIN Break Detection Clear Flag
  - UART\_CLEAR\_CTSF: CTS Interrupt Clear Flag
  - UART\_CLEAR\_RTOF: Receiver Time Out Clear Flag

- UART\_CLEAR\_EOBF: End Of Block Clear Flag
- UART\_CLEAR\_CMF: Character Match Clear Flag

**Return value:**

- None

`__HAL_UART_CLEAR_PEFLAG`

**Description:**

- Clear the UART PE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_CLEAR_FEFLAG`

**Description:**

- Clear the UART FE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_CLEAR_NEFLAG`

**Description:**

- Clear the UART NE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_CLEAR_OREFLAG`

**Description:**

- Clear the UART ORE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_CLEAR_IDLEFLAG`

**Description:**

- Clear the UART IDLE pending flag.

**Parameters:**

- `__HANDLE__`: specifies the UART

`__HAL_UART_GET_FLAG`

Handle.

**Return value:**

- None

**Description:**

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

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
  - `UART_FLAG_REACK`: Receive enable acknowledge flag
  - `UART_FLAG_TEACK`: Transmit enable acknowledge flag
  - `UART_FLAG_WUF`: Wake up from stop mode flag
  - `UART_FLAG_RWU`: Receiver wake up flag (is the UART in mute mode)
  - `UART_FLAG_SBKF`: Send Break flag
  - `UART_FLAG_CMF`: Character match flag
  - `UART_FLAG_BUSY`: Busy flag
  - `UART_FLAG_ABRF`: Auto Baud rate detection flag
  - `UART_FLAG_ABRE`: Auto Baud rate detection error flag
  - `UART_FLAG_EOBF`: End of block flag
  - `UART_FLAG_RTOF`: Receiver timeout flag
  - `UART_FLAG_CTS`: CTS Change flag (not available for UART4 and UART5)
  - `UART_FLAG_LBD`: LIN Break detection flag
  - `UART_FLAG_TXE`: Transmit data register empty flag
  - `UART_FLAG_TC`: Transmission Complete flag
  - `UART_FLAG_RXNE`: Receive data register not empty flag
  - `UART_FLAG_IDLE`: Idle Line detection flag
  - `UART_FLAG_ORE`: OverRun Error flag
  - `UART_FLAG_NE`: Noise Error flag
  - `UART_FLAG_FE`: Framing Error flag

- UART\_FLAG\_PE: Parity Error flag

**Return value:**

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

**Description:**

- Enables the specified UART interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle.
- \_\_INTERRUPT\_\_: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART\_IT\_WUF: Wakeup from stop mode interrupt
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty interrupt
  - UART\_IT\_IDLE: Idle line detection interrupt
  - UART\_IT\_PE: Parity Error interrupt
  - UART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

**Description:**

- Disables the specified UART interrupt.

**Parameters:**

- \_\_HANDLE\_\_: specifies the UART Handle.
- \_\_INTERRUPT\_\_: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt
  - UART\_IT\_LBD: LIN Break detection

`__HAL_UART_ENABLE_IT``__HAL_UART_DISABLE_IT`

- 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

**Description:**

- Checks whether the specified UART interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.
- `__IT__`: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART\_IT\_CM: Character match interrupt
  - UART\_IT\_CTS: CTS change interrupt (not available for UART4 and UART5)
  - UART\_IT\_LBD: LIN Break detection interrupt
  - UART\_IT\_TXE: Transmit Data Register empty interrupt
  - UART\_IT\_TC: Transmission complete interrupt
  - UART\_IT\_RXNE: Receive Data register not empty interrupt
  - UART\_IT\_IDLE: Idle line detection interrupt
  - UART\_IT\_ORE: OverRun Error interrupt
  - UART\_IT\_NE: Noise Error interrupt
  - UART\_IT\_FE: Framing Error interrupt
  - UART\_IT\_PE: Parity Error interrupt

**Return value:**

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

`__HAL_UART_GET_IT``__HAL_UART_GET_IT_SOURCE`**Description:**

- Checks whether the specified UART interrupt source is enabled.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.
- `__IT__`: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - `UART_IT_CTS`: CTS change interrupt (not available for UART4 and UART5)
  - `UART_IT_LBD`: LIN Break detection interrupt
  - `UART_IT_TXE`: Transmit Data Register empty interrupt
  - `UART_IT_TC`: Transmission complete interrupt
  - `UART_IT_RXNE`: Receive Data register not empty interrupt
  - `UART_IT_IDLE`: Idle line detection interrupt
  - `UART_IT_ORE`: OverRun Error interrupt
  - `UART_IT_NE`: Noise Error interrupt
  - `UART_IT_FE`: Framing Error interrupt
  - `UART_IT_PE`: Parity Error interrupt

**Return value:**

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

`__HAL_UART_SEND_REQ`**Description:**

- Set a specific UART request flag.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.
- `__REQ__`: specifies the request flag to set This parameter can be one of the following values:
  - `UART_AUTOBAUD_REQUEST`: Auto-Baud Rate Request
  - `UART_SENDBREAK_REQUEST`: Send Break Request
  - `UART_MUTE_MODE_REQUEST`: Mute Mode Request
  - `UART_RXDATA_FLUSH_REQUEST`: Receive Data flush Request
  - `UART_TXDATA_FLUSH_REQUEST`: Transmit data flush Request

**Return value:**

`__HAL_UART_ONE_BIT_SAMPLE_ENABLE`

- None

**Description:**

- Enables the UART one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_ONE_BIT_SAMPLE_DISABLE`

**Description:**

- Disables the UART one bit sample method.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_ENABLE`

**Description:**

- Enable UART.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_DISABLE`

**Description:**

- Disable UART.

**Parameters:**

- `__HANDLE__`: specifies the UART Handle.

**Return value:**

- None

`__HAL_UART_HWCONTROL_CTS_ENABLE`

**Description:**

- Enable CTS flow control 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 : 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_HWCONTROL_CTS_DISABLE`

**Description:**

- Disable CTS flow control This macro allows to disable CTS hardware flow control for a given USART instance, without need to call

**Parameters:**

- `__HANDLE__`: specifies the USART 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 : 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_HWCONTROL_RTS_ENABLE`

**Description:**

- Enable RTS flow control This macro allows to enable RTS hardware flow control for a given USART 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 : UART instance should have already been initialised (through call of `HAL_UART_Init()`) macro could only be called when corresponding UART instance is disabled (i.e `__HAL_UART_DISABLE(__HANDLE__)`) and should be followed by an Enable macro (i.e `__HAL_UART_ENABLE(__HANDLE__)`).

**UART Status Flags**

UART\_FLAG\_TEACK

UART\_FLAG\_SBKF

UART\_FLAG\_CMF

UART\_FLAG\_BUSY

UART\_FLAG\_ABRF

UART\_FLAG\_ABRE

UART\_FLAG\_EOBF

UART\_FLAG\_RTOF

UART\_FLAG\_CTS

UART\_FLAG\_CTSIF

UART\_FLAG\_LBDF

UART\_FLAG\_TXE

UART\_FLAG\_TC

UART\_FLAG\_RXNE

UART\_FLAG\_IDLE

UART\_FLAG\_ORE

UART\_FLAG\_NE

UART\_FLAG\_FE

UART\_FLAG\_PE

***UART Half Duplex Selection***

UART\_HALF\_DUPLEX\_DISABLE

UART\_HALF\_DUPLEX\_ENABLE

***UART Hardware Flow Control***

UART\_HWCONTROL\_NONE

UART\_HWCONTROL\_RTS

UART\_HWCONTROL\_CTS

UART\_HWCONTROL\_RTS\_CTS

***UART Interrupts Flag Mask***

UART\_IT\_MASK

***UART Interrupts Definition***

UART\_IT\_PE

UART\_IT\_TXE

UART\_IT\_TC

UART\_IT\_RXNE

UART\_IT\_IDLE

UART\_IT\_LBD

UART\_IT\_CTS

UART\_IT\_CM

UART\_IT\_ERR

UART\_IT\_ORE

UART\_IT\_NE

UART\_IT\_FE

**UART Interruption Clear Flags**

UART\_CLEAR\_PEF      Parity Error Clear Flag

UART\_CLEAR\_FEF      Framing Error Clear Flag

UART\_CLEAR\_NEF      Noise detected Clear Flag

UART\_CLEAR\_OREF      OverRun Error Clear Flag

UART\_CLEAR\_IDLEF      IDLE line detected Clear Flag

UART\_CLEAR\_TCF      Transmission Complete Clear Flag

UART\_CLEAR\_LBDF      LIN Break Detection Clear Flag

UART\_CLEAR\_CTSF      CTS Interrupt Clear Flag

UART\_CLEAR\_RTOF      Receiver Time Out Clear Flag

UART\_CLEAR\_EOBF      End Of Block Clear Flag

UART\_CLEAR\_CMF      Character Match Clear Flag

**UART Local Interconnection Network mode**

UART\_LIN\_DISABLE

UART\_LIN\_ENABLE

**UART LIN Break Detection**

UART\_LINBREAKDETECTLENGTH\_10B

UART\_LINBREAKDETECTLENGTH\_11B

**UART Transfer Mode**

UART\_MODE\_RX

UART\_MODE\_TX

UART\_MODE\_TX\_RX

**UART Advanced Feature MSB First**

UART\_ADVFEATURE\_MSBFIRST\_DISABLE

UART\_ADVFEATURE\_MSBFIRST\_ENABLE

**UART Advanced Feature Mute Mode Enable**

UART\_ADVFEATURE\_MUTEMODE\_DISABLE

UART\_ADVFEATURE\_MUTEMODE\_ENABLE

**UART One Bit Sampling Method**

UART\_ONE\_BIT\_SAMPLE\_DISABLE

UART\_ONE\_BIT\_SAMPLE\_ENABLE

**UART Advanced Feature Overrun Disable**

UART\_ADVFEATURE\_OVERRUN\_ENABLE

UART\_ADVFEATURE\_OVERRUN\_DISABLE

**UART Over Sampling**

UART\_OVERSAMPLING\_16

UART\_OVERSAMPLING\_8

**UART Parity**

UART\_PARITY\_NONE

UART\_PARITY\_EVEN

UART\_PARITY\_ODD

**UART Private Macros**

UART\_DIV\_LPUART

**Description:**

- BRR division operation to set BRR register with LPUART.

**Parameters:**

- `_PCLK_`: LPUART clock
- `_BAUD_`: Baud rate set by the user

**Return value:**

- Division: result

UART\_DIV\_SAMPLING8

**Description:**

- BRR division operation to set BRR register in 8-bit oversampling mode.

**Parameters:**

- `_PCLK_`: UART clock
- `_BAUD_`: Baud rate set by the user

**Return value:**

- Division: result

UART\_DIV\_SAMPLING16

**Description:**

- BRR division operation to set BRR register in 16-bit oversampling mode.

**Parameters:**

- `_PCLK_`: UART clock
- `_BAUD_`: Baud rate set by the user

**Return value:**

**IS\_UART\_BAUDRATE**

- Division: result

**Description:**

- Check UART Baud rate.

**Parameters:**

- BAUDRATE: Baudrate specified by the user The maximum Baud Rate is derived from the maximum clock on F7 (i.e. 216 MHz) divided by the smallest oversampling used on the USART (i.e. 8)

**Return value:**

- Test: result (TRUE or FALSE).

**IS\_UART\_ASSERTIONTIME****Description:**

- Check UART assertion time.

**Parameters:**

- TIME: 5-bit value assertion time

**Return value:**

- Test: result (TRUE or FALSE).

**IS\_UART\_DEASSERTIONTIME****Description:**

- Check UART deassertion time.

**Parameters:**

- TIME: 5-bit value deassertion time

**Return value:**

- Test: result (TRUE or FALSE).

**IS\_UART\_STOPBITS****IS\_UART\_PARITY****IS\_UART\_HARDWARE\_FLOW\_CONTROL****IS\_UART\_MODE****IS\_UART\_STATE****IS\_UART\_OVERSAMPLING****IS\_UART\_ONE\_BIT\_SAMPLE****IS\_UART\_ADVFEATURE\_AUTOBAUDRATEMODE****IS\_UART\_RECEIVER\_TIMEOUT****IS\_UART\_LIN****IS\_UART\_WAKEUPMETHOD****IS\_UART\_LIN\_BREAK\_DETECT\_LENGTH**

IS\_UART\_DMA\_TX  
IS\_UART\_DMA\_RX  
IS\_UART\_HALF\_DUPLEX  
IS\_UART\_REQUEST\_PARAMETER  
IS\_UART\_ADVFEATURE\_INIT  
IS\_UART\_ADVFEATURE\_TXINV  
IS\_UART\_ADVFEATURE\_RXINV  
IS\_UART\_ADVFEATURE\_DATAINV  
IS\_UART\_ADVFEATURE\_SWAP  
IS\_UART\_OVERRUN  
IS\_UART\_ADVFEATURE\_AUTOBAUDRATE  
IS\_UART\_ADVFEATURE\_DMAONRXERROR  
IS\_UART\_ADVFEATURE\_MSBFIRST  
IS\_UART\_MUTE\_MODE  
IS\_UART\_DE\_POLARITY

***UART Receiver TimeOut***

UART\_RECEIVER\_TIMEOUT\_DISABLE  
UART\_RECEIVER\_TIMEOUT\_ENABLE

***UART Request Parameters***

UART_AUTOBAUD_REQUEST	Auto-Baud Rate Request
UART_SENDBREAK_REQUEST	Send Break Request
UART_MUTE_MODE_REQUEST	Mute Mode Request
UART_RXDATA_FLUSH_REQUEST	Receive Data flush Request
UART_TXDATA_FLUSH_REQUEST	Transmit data flush Request

***UART Advanced Feature RX Pin Active Level Inversion***

UART\_ADVFEATURE\_RXINV\_DISABLE  
UART\_ADVFEATURE\_RXINV\_ENABLE

***UART Advanced Feature RX TX Pins Swap***

UART\_ADVFEATURE\_SWAP\_DISABLE  
UART\_ADVFEATURE\_SWAP\_ENABLE

***UART State***

UART\_STATE\_DISABLE  
UART\_STATE\_ENABLE

***UART Number of Stop Bits***

UART\_STOPBITS\_1  
UART\_STOPBITS\_2

***UART polling-based communications time-out value***

HAL\_UART\_TIMEOUT\_VALUE

***UART Advanced Feature TX Pin Active Level Inversion***

UART\_ADVFEATURE\_TXINV\_DISABLE

UART\_ADVFEATURE\_TXINV\_ENABLE

***UART WakeUp Methods***

UART\_WAKEUPMETHOD\_IDLELINE

UART\_WAKEUPMETHOD\_ADDRESSMARK



## 60 HAL UART Extension Driver

### 60.1 UARTEEx Firmware driver defines

#### 60.1.1 UARTEEx

##### *UARTEEx Exported Macros*

##### UART\_GETCLOCKSOURCE

##### Description:

- Reports the UART clock source.

##### Parameters:

- `__HANDLE__`: specifies the UART Handle
- `__CLOCKSOURCE__`: output variable

##### Return value:

- UART: clocking source, written in `__CLOCKSOURCE__`.

##### UART\_MASK\_COMPUTATION

##### Description:

- Reports the UART mask to apply to retrieve the received data according to the word length and to the parity bits activation.

##### Parameters:

- `__HANDLE__`: specifies the UART Handle

##### Return value:

- mask: to apply to UART RDR register value.

##### *UARTEEx WakeUp Address Length*

UART\_ADDRESS\_DETECT\_4B

UART\_ADDRESS\_DETECT\_7B

IS\_UART\_ADDRESSELENGTH\_DETECT

##### *UARTEEx Word Length*

UART\_WORDLENGTH\_7B

UART\_WORDLENGTH\_8B

UART\_WORDLENGTH\_9B

IS\_UART\_WORD\_LENGTH

IS\_LIN\_WORD\_LENGTH

## 61 HAL USART Generic Driver

### 61.1 USART Firmware driver registers structures

#### 61.1.1 USART\_InitTypeDef

##### Data Fields

- *uint32\_t BaudRate*
- *uint32\_t WordLength*
- *uint32\_t StopBits*
- *uint32\_t Parity*
- *uint32\_t Mode*
- *uint32\_t OverSampling*
- *uint32\_t CLKPolarity*
- *uint32\_t CLKPhase*
- *uint32\_t CLKLastBit*

##### Field Documentation

- ***uint32\_t USART\_InitTypeDef::BaudRate***  
This member configures the Usart communication baud rate. The baud rate is computed using the following formula: Baud Rate Register = ((PCLKx) / ((huart->Init.BaudRate)))
- ***uint32\_t USART\_InitTypeDef::WordLength***  
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [USARTEx\\_Word\\_Length](#)
- ***uint32\_t USART\_InitTypeDef::StopBits***  
Specifies the number of stop bits transmitted. This parameter can be a value of [USART\\_Stop\\_Bits](#)
- ***uint32\_t USART\_InitTypeDef::Parity***  
Specifies the parity mode. This parameter can be a value of [USART\\_Parity](#)  
**Note:**When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32\_t USART\_InitTypeDef::Mode***  
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [USART\\_Mode](#)
- ***uint32\_t USART\_InitTypeDef::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 [USART\\_Over\\_Sampling](#)
- ***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](#)

### 61.1.2 USART\_HandleTypeDef

#### Data Fields

- ***USART\_TypeDef \* Instance***
- ***USART\_InitTypeDef Init***
- ***uint8\_t \* pTxBuffPtr***
- ***uint16\_t TxXferSize***
- ***uint16\_t TxXferCount***
- ***uint8\_t \* pRxBuffPtr***
- ***uint16\_t RxXferSize***
- ***uint16\_t RxXferCount***
- ***uint16\_t Mask***
- ***DMA\_HandleTypeDef \* hdmatx***
- ***DMA\_HandleTypeDef \* hdmarx***
- ***HAL\_LockTypeDef Lock***
- ***HAL\_USART\_StateTypeDef State***
- ***\_\_IO uint32\_t ErrorCode***

#### Field Documentation

- ***USART\_TypeDef\* USART\_HandleTypeDef::Instance***  
USART registers base address
- ***USART\_InitTypeDef USART\_HandleTypeDef::Init***  
USART communication parameters
- ***uint8\_t\* USART\_HandleTypeDef::pTxBuffPtr***  
Pointer to USART Tx transfer Buffer
- ***uint16\_t USART\_HandleTypeDef::TxXferSize***  
USART Tx Transfer size
- ***uint16\_t USART\_HandleTypeDef::TxXferCount***  
USART Tx Transfer Counter
- ***uint8\_t\* USART\_HandleTypeDef::pRxBuffPtr***  
Pointer to USART Rx transfer Buffer
- ***uint16\_t USART\_HandleTypeDef::RxXferSize***  
USART Rx Transfer size
- ***uint16\_t USART\_HandleTypeDef::RxXferCount***  
USART Rx Transfer Counter
- ***uint16\_t USART\_HandleTypeDef::Mask***  
USART Rx RDR register mask
- ***DMA\_HandleTypeDef\* USART\_HandleTypeDef::hdmatx***  
USART Tx DMA Handle parameters
- ***DMA\_HandleTypeDef\* USART\_HandleTypeDef::hdmarx***  
USART Rx DMA Handle parameters
- ***HAL\_LockTypeDef USART\_HandleTypeDef::Lock***  
Locking object
- ***HAL\_USART\_StateTypeDef USART\_HandleTypeDef::State***  
USART communication state

- `__IO uint32_t USART_HandleTypeDef::ErrorCode`  
USART Error code

## 61.2 USART Firmware driver API description

### 61.2.1 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 implement 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.
    - 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.
  - d. DMA Configuration if you need to use DMA process (`HAL_USART_Transmit_DMA()`, `HAL_USART_Receive_IT()` and `HAL_USART_TransmitReceive_IT()` APIs):
    - Declare a DMA handle structure for the Tx/Rx stream.
    - Enable the DMAx interface clock.
    - Configure the declared DMA handle structure with the required Tx/Rx parameters.
    - Configure the DMA Tx/Rx Stream.
    - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
    - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the `husart Init` structure.
4. Initialize the USART registers by calling the `HAL_USART_Init()` API:
  - These API's configures also the low level Hardware (GPIO, CLOCK, CORTEX...etc) by calling the customized `HAL_USART_MspInit(&husart)` API.

### 61.2.2 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 M1 and M0 bits (7-bit, 8-bit or 9-bit), the possible USART frame formats are as listed in [Table 16: "USART frame formats"](#).

- USART polarity
- USART phase
- USART LastBit
- Receiver/transmitter modes

**Table 17: USART frame formats**

M1M0 bits	PCE bit	USART frame
10	0	SB   7-bit data   STB
10	1	SB   6-bit data   PB   STB

The HAL\_USART\_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- [HAL\\_USART\\_Init\(\)](#)
- [HAL\\_USART\\_DeInit\(\)](#)
- [HAL\\_USART\\_MspInit\(\)](#)
- [HAL\\_USART\\_MspDeInit\(\)](#)
- [HAL\\_USART\\_CheckIdleState\(\)](#)

### 61.2.3 IO operation functions

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 mode of transfer:
  - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL\_USART\_TxCpltCallback(), HAL\_USART\_RxCpltCallback() and HAL\_USART\_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL\_USART\_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode API's are :
  - HAL\_USART\_Transmit() in simplex mode
  - HAL\_USART\_Receive() in full duplex receive only
  - HAL\_USART\_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
  - HAL\_USART\_Transmit\_IT() in simplex mode
  - HAL\_USART\_Receive\_IT() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_IT() in full duplex mode
  - HAL\_USART\_IRQHandler()
4. No-Blocking mode functions with DMA are :
  - HAL\_USART\_Transmit\_DMA() in simplex mode
  - HAL\_USART\_Receive\_DMA() in full duplex receive only
  - HAL\_USART\_TransmitReceive\_DMA() in full duplex mode
  - HAL\_USART\_DMAPause()
  - HAL\_USART\_DMAResume()

- HAL\_USART\_DMAStop()
- 5. A set of Transfer Complete Callbacks are provided in No\_Blocking mode:
  - HAL\_USART\_TxCpltCallback()
  - HAL\_USART\_RxCpltCallback()
  - HAL\_USART\_TxHalfCpltCallback()
  - HAL\_USART\_RxHalfCpltCallback()
  - HAL\_USART\_ErrorCallback()
  - HAL\_USART\_TxRxCpltCallback()

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\(\)](#)

#### 61.2.4 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\(\)](#)

#### 61.2.5 HAL\_USART\_Init

Function Name	<b>HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)</b>
Function Description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

**61.2.6 HAL\_USART\_DeInit**

Function Name **HAL\_StatusTypeDef HAL\_USART\_DeInit (USART\_HandleTypeDef \* husart)**

Function Description DeInitializes the USART peripheral.

Parameters

- **husart:** USART handle

Return values

- HAL status

**61.2.7 HAL\_USART\_MspInit**

Function Name **void HAL\_USART\_MspInit (USART\_HandleTypeDef \* husart)**

Function Description USART MSP Init.

Parameters

- **husart:** USART handle

Return values

- None

**61.2.8 HAL\_USART\_MspDeInit**

Function Name **void HAL\_USART\_MspDeInit (USART\_HandleTypeDef \* husart)**

Function Description USART MSP DeInit.

Parameters

- **husart:** USART handle

Return values

- None

**61.2.9 HAL\_USART\_CheckIdleState**

Function Name **HAL\_StatusTypeDef HAL\_USART\_CheckIdleState (USART\_HandleTypeDef \* husart)**

Function Description

**61.2.10 HAL\_USART\_Transmit**

Function Name **HAL\_StatusTypeDef HAL\_USART\_Transmit (USART\_HandleTypeDef \* husart, uint8\_t \* pTxData, uint16\_t Size, uint32\_t Timeout)**

Function Description Simplex Send an amount of data in blocking mode.

Parameters

- **husart:** USART handle
- **pTxData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** : Timeout duration

Return values

- HAL status

**61.2.11 HAL\_USART\_Receive**

Function Name **HAL\_StatusTypeDef HAL\_USART\_Receive (USART\_HandleTypeDef \* husart, uint8\_t \* pRxData, uint16\_t Size, uint32\_t Timeout)**

Function Description Receive an amount of data in blocking mode.

Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</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>
Notes	<ul style="list-style-type: none"> <li>• To receive synchronous data, dummy data are simultaneously transmitted</li> </ul>

### 61.2.12 HAL\_USART\_TransmitReceive

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive</b> (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Full-Duplex Send and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</li> <li>• <b>pTxData</b>: pointer to TX data buffer</li> <li>• <b>pRxData</b>: pointer to RX data buffer</li> <li>• <b>Size</b>: amount of data to be sent (same amount 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>

### 61.2.13 HAL\_USART\_Transmit\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit_IT</b> (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</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>

### 61.2.14 HAL\_USART\_Receive\_IT

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive_IT</b> (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</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>

### 61.2.15 HAL\_USART\_TransmitReceive\_IT



Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive_IT</b> (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Send and Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</li> <li>• <b>pTxData</b>: pointer to TX data buffer</li> <li>• <b>pRxData</b>: pointer to RX data buffer</li> <li>• <b>Size</b>: amount of data to be sent (same amount to be received)</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL status</li> </ul>

### 61.2.16 HAL\_USART\_Transmit\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_USART_Transmit_DMA</b> (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</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>

### 61.2.17 HAL\_USART\_Receive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_USART_Receive_DMA</b> (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</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>• When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)</li> <li>• The USART DMA transmit stream must be configured in order to generate the clock for the slave.</li> </ul>

### 61.2.18 HAL\_USART\_TransmitReceive\_DMA

Function Name	<b>HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA</b> (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Transmit Receive an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</li> <li>• <b>pTxData</b>: pointer to TX data buffer</li> <li>• <b>pRxData</b>: pointer to RX data buffer</li> </ul>

	<ul style="list-style-type: none"><li>• <b>Size:</b> amount of data to be received/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 USART parity is enabled (PCE = 1) the data received contain the parity bit.</li></ul>

### 61.2.19 HAL\_USART\_DMAPause

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)</b>
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>husart:</b> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 61.2.20 HAL\_USART\_DMAResume

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)</b>
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>husart:</b> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 61.2.21 HAL\_USART\_DMAStop

Function Name	<b>HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)</b>
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"><li>• <b>husart:</b> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 61.2.22 HAL\_USART\_IRQHandler

Function Name	<b>void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)</b>
Function Description	This function handles USART interrupt request.
Parameters	<ul style="list-style-type: none"><li>• <b>husart:</b> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

### 61.2.23 HAL\_USART\_TxCpltCallback

Function Name	<b>void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)</b>
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"><li>• <b>husart:</b> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**61.2.24 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> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**61.2.25 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> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**61.2.26 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> usart handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**61.2.27 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> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**61.2.28 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> USART handle</li></ul>
Return values	<ul style="list-style-type: none"><li>• None</li></ul>

**61.2.29 HAL\_USART\_GetState**

Function Name	<b>HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)</b>
Function Description	return the USART state

Parameters	<ul style="list-style-type: none"> <li>• <b>husart</b>: USART handle</li> </ul>
Return values	<ul style="list-style-type: none"> <li>• HAL state</li> </ul>

### 61.2.30 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>

## 61.3 USART Firmware driver defines

### 61.3.1 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**

<b>__HAL_USART_RESET_HANDLE_STATE</b>	<b>Description:</b>
ATE	<ul style="list-style-type: none"> <li>• Reset USART handle state.</li> </ul>
	<b>Parameters:</b>
	<ul style="list-style-type: none"> <li>• <b>__HANDLE__</b>: USART handle.</li> </ul>
	<b>Return value:</b>
	<ul style="list-style-type: none"> <li>• None</li> </ul>

---

**\_\_HAL\_USART\_GET\_FLAG****Description:**

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

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle
- **\_\_FLAG\_\_**: specifies the flag to check. This parameter can be one of the following values:
  - USART\_FLAG\_REACK: Receive enable acknowledge flag
  - USART\_FLAG\_TEACK: Transmit enable acknowledge flag
  - USART\_FLAG\_BUSY: Busy flag
  - USART\_FLAG\_CTS: CTS Change flag
  - USART\_FLAG\_TXE: Transmit data register empty flag
  - USART\_FLAG\_TC: Transmission Complete flag
  - USART\_FLAG\_RXNE: Receive data register not empty flag
  - USART\_FLAG\_IDLE: Idle Line detection flag
  - USART\_FLAG\_ORE: OverRun Error flag
  - USART\_FLAG\_NE: Noise Error flag
  - USART\_FLAG\_FE: Framing Error flag
  - USART\_FLAG\_PE: Parity Error flag

**Return value:**

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

**\_\_HAL\_USART\_ENABLE\_IT****Description:**

- Enables the specified USART interrupt.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle
- **\_\_INTERRUPT\_\_**: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_PE: Parity Error interrupt
  - USART\_IT\_ERR: Error interrupt(Frame error, noise error, overrun error)

`__HAL_USART_DISABLE_IT`**Return value:**

- None

**Description:**

- Disables the specified USART interrupt.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle.
- `__INTERRUPT__`: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  - `USART_IT_TXE`: Transmit Data Register empty interrupt
  - `USART_IT_TC`: Transmission complete interrupt
  - `USART_IT_RXNE`: Receive Data register not empty interrupt
  - `USART_IT_IDLE`: Idle line detection interrupt
  - `USART_IT_PE`: Parity Error interrupt
  - `USART_IT_ERR`: Error interrupt(Frame error, noise error, overrun error)

**Return value:**

- None

`__HAL_USART_GET_IT`**Description:**

- Checks whether the specified USART interrupt has occurred or not.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle
- `__IT__`: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - `USART_IT_TXE`: Transmit Data Register empty interrupt
  - `USART_IT_TC`: Transmission complete interrupt
  - `USART_IT_RXNE`: Receive Data register not empty interrupt
  - `USART_IT_IDLE`: Idle line detection interrupt
  - `USART_IT_ORE`: OverRun Error interrupt
  - `USART_IT_NE`: Noise Error interrupt
  - `USART_IT_FE`: Framing Error interrupt
  - `USART_IT_PE`: Parity Error interrupt

**Return value:**

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

---

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

- Checks whether the specified USART interrupt source is enabled.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle.
- **\_\_IT\_\_**: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART\_IT\_TXE: Transmit Data Register empty interrupt
  - USART\_IT\_TC: Transmission complete interrupt
  - USART\_IT\_RXNE: Receive Data register not empty interrupt
  - USART\_IT\_IDLE: Idle line detection interrupt
  - USART\_IT\_ORE: OverRun Error interrupt
  - USART\_IT\_NE: Noise Error interrupt
  - USART\_IT\_FE: Framing Error interrupt
  - USART\_IT\_PE: Parity Error interrupt

**Return value:**

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

---

**\_\_HAL\_USART\_CLEAR\_IT****Description:**

- Clears the specified USART ISR flag, in setting the proper ICR register flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle.
- **\_\_IT\_CLEAR\_\_**: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - USART\_CLEAR\_PEF: Parity Error Clear Flag
  - USART\_CLEAR\_FEF: Framing Error Clear Flag
  - USART\_CLEAR\_NEF: Noise detected Clear Flag
  - USART\_CLEAR\_OREF: OverRun Error Clear Flag
  - USART\_CLEAR\_IDLEF: IDLE line detected Clear Flag
  - USART\_CLEAR\_TCF: Transmission Complete Clear Flag
  - USART\_CLEAR\_CTSF: CTS Interrupt Clear Flag

**Return value:**

**\_\_HAL\_USART\_SEND\_REQ**

- None

**Description:**

- Set a specific USART request flag.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle.
- **\_\_REQ\_\_**: specifies the request flag to set  
This parameter can be one of the following values:
  - USART\_RXDATA\_FLUSH\_REQUEST : Receive Data flush Request
  - USART\_TXDATA\_FLUSH\_REQUEST: Transmit data flush Request

**Return value:**

- None

**Description:**

- Enable USART.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle.

**Return value:**

- None

**Description:**

- Disable USART.

**Parameters:**

- **\_\_HANDLE\_\_**: specifies the USART Handle.

**Return value:**

- None

**\_\_HAL\_USART\_ENABLE****\_\_HAL\_USART\_DISABLE****USART Flags**

USART\_FLAG\_REACK

USART\_FLAG\_TEACK

USART\_FLAG\_BUSY

USART\_FLAG\_CTS

USART\_FLAG\_CTSIF

USART\_FLAG\_LBDF

USART\_FLAG\_TXE

USART\_FLAG\_TC

USART\_FLAG\_RXNE



USART\_FLAG\_IDLE

USART\_FLAG\_ORE

USART\_FLAG\_NE

USART\_FLAG\_FE

USART\_FLAG\_PE

***USART Interrupts Definition***

USART\_IT\_PE

USART\_IT\_TXE

USART\_IT\_TC

USART\_IT\_RXNE

USART\_IT\_IDLE

USART\_IT\_ERR

USART\_IT\_ORE

USART\_IT\_NE

USART\_IT\_FE

***USART Interruption Clear Flags***

USART\_CLEAR\_PEF      Parity Error Clear Flag

USART\_CLEAR\_FEF      Framing Error Clear Flag

USART\_CLEAR\_NEF      Noise detected Clear Flag

USART\_CLEAR\_OREF      OverRun Error Clear Flag

USART\_CLEAR\_IDLEF    IDLE line detected Clear Flag

USART\_CLEAR\_TCF      Transmission Complete Clear Flag

USART\_CLEAR\_CTSF    CTS Interrupt Clear Flag

***USART Last Bit***

USART\_LASTBIT\_DISABLE

USART\_LASTBIT\_ENABLE

***USART Mode***

USART\_MODE\_RX

USART\_MODE\_TX

USART\_MODE\_TX\_RX

***USART Over Sampling***

USART\_OVERSAMPLING\_16

USART\_OVERSAMPLING\_8

***USART Parity***

USART\_PARITY\_NONE

USART\_PARITY\_EVEN

USART\_PARITY\_ODD

**USART Private Constants**

DUMMY\_DATA

TEACK\_REACK\_TIMEOUT

USART\_TXDMA\_TIMEOUTVALUE

USART\_TIMEOUT\_VALUE

USART\_CR1\_FIELDS

USART\_CR2\_FIELDS

USART\_IT\_MASK

**USART Private Macros**

USART\_GETCLOCKSOURCE

**Description:**

- Reports the USART clock source.

**Parameters:**

- `__HANDLE__`: specifies the USART Handle
- `__CLOCKSOURCE__`: : output variable

**Return value:**

- the: USART clocking source, written in `__CLOCKSOURCE__`.

IS\_USART\_STOPBITS

IS\_USART\_PARITY

IS\_USART\_MODE

IS\_USART\_OVERSAMPLING

IS\_USART\_CLOCK

IS\_USART\_POLARITY

IS\_USART\_PHASE

IS\_USART\_LASTBIT

IS\_USART\_REQUEST\_PARAMETER

IS\_USART\_BAUDRATE

**USART Request Parameters**

USART\_RXDATA\_FLUSH\_REQUEST    Receive Data flush Request

USART\_TXDATA\_FLUSH\_REQUEST    Transmit data flush Request

**USART Number of Stop Bits**

USART\_STOPBITS\_1

USART\_STOPBITS\_2

USART\_STOPBITS\_1\_5

## 62 HAL USART Extension Driver

### 62.1 USARTEEx Firmware driver defines

#### 62.1.1 USARTEEx

##### *USARTEEx Private Macros*

`__HAL_USART_MASK_COMPUTATION` **Description:**

- Computes the USART mask to apply to retrieve the received data according to the word length and to the parity bits activation.

##### **Parameters:**

- `__HANDLE__`: specifies the USART Handle

##### **Return value:**

- none

`IS_USART_WORD_LENGTH`

##### *USARTEEx Word Length*

`USART_WORDLENGTH_7B`

`USART_WORDLENGTH_8B`

`USART_WORDLENGTH_9B`

## 63 HAL WWDG Generic Driver

### 63.1 WWDG Firmware driver registers structures

#### 63.1.1 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

#### 63.1.2 WWDG\_HandleTypeDef

##### Data Fields

- *WWDG\_TypeDef \* Instance*
- *WWDG\_InitTypeDef Init*
- *HAL\_LockTypeDef Lock*
- *\_\_IO HAL\_WWDG\_StateTypeDef State*

##### Field Documentation

- *WWDG\_TypeDef\* WWDG\_HandleTypeDef::Instance*  
Register base address
- *WWDG\_InitTypeDef WWDG\_HandleTypeDef::Init*  
WWDG required parameters
- *HAL\_LockTypeDef WWDG\_HandleTypeDef::Lock*  
WWDG locking object
- *\_\_IO HAL\_WWDG\_StateTypeDef WWDG\_HandleTypeDef::State*  
WWDG communication state

## 63.2 WWDG Firmware driver API description

### 63.2.1 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.
- WWDGRST flag in RCC\_CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- $\text{WWDG clock (Hz)} = \text{PCLK1} / (4096 * \text{Prescaler})$
- $\text{WWDG timeout (mS)} = 1000 * \text{Counter} / \text{WWDG clock}$
- WWDG Counter refresh is allowed between the following limits :
  - $\text{min time (mS)} = 1000 * (\text{Counter} - \text{Window}) / \text{WWDG clock}$
  - $\text{max time (mS)} = 1000 * (\text{Counter} - 0x40) / \text{WWDG clock}$
- Min-max timeout value at 50 MHz(PCLK1): 81.9 us / 41.9 ms

### 63.2.2 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

### 63.2.3 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\(\)](#)

### 63.2.4 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\(\)](#)

### 63.2.5 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\(\)](#)

### 63.2.6 HAL\_WWDG\_Init

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Init</b> (WWDG_HandleTypeDef * hwwdg)
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>hwwdg</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>

### 63.2.7 HAL\_WWDG\_DeInit

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_DeInit</b> (WWDG_HandleTypeDef * hwwdg)
Function Description	DeInitializes the WWDG peripheral.
Parameters	<ul style="list-style-type: none"> <li>• <b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified</li> </ul>

WWDG module.

Return values

- HAL status

### 63.2.8 HAL\_WWDG\_MspInit

Function Name **void HAL\_WWDG\_MspInit (WWDG\_HandleTypeDef \* hwwdg)**

Function Description Initializes the WWDG MSP.

Parameters

- **hwwdg**: pointer to a WWDG\_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- None

### 63.2.9 HAL\_WWDG\_MspDeInit

Function Name **void HAL\_WWDG\_MspDeInit (WWDG\_HandleTypeDef \* hwwdg)**

Function Description DeInitializes the WWDG MSP.

Parameters

- **hwwdg**: pointer to a WWDG\_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- None

### 63.2.10 HAL\_WWDG\_WakeupCallback

Function Name **void HAL\_WWDG\_WakeupCallback (WWDG\_HandleTypeDef \* hwwdg)**

Function Description Early Wakeup WWDG callback.

Parameters

- **hwwdg**: pointer to a WWDG\_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- None

### 63.2.11 HAL\_WWDG\_Start

Function Name **HAL\_StatusTypeDef HAL\_WWDG\_Start (WWDG\_HandleTypeDef \* hwwdg)**

Function Description Starts the WWDG.

Parameters

- **hwwdg**: pointer to a WWDG\_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- HAL status

### 63.2.12 HAL\_WWDG\_Start\_IT

Function Name **HAL\_StatusTypeDef HAL\_WWDG\_Start\_IT (WWDG\_HandleTypeDef \* hwwdg)**

Function Description Starts the WWDG with interrupt enabled.

Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</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>

### 63.2.13 HAL\_WWDG\_Refresh

Function Name	<b>HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg, uint32_t Counter)</b>
Function Description	Refreshes the WWDG.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</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>

### 63.2.14 HAL\_WWDG\_IRQHandler

Function Name	<b>void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)</b>
Function Description	Handles WWDG interrupt request.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</b>: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.</li> </ul>
Return values	<ul style="list-style-type: none"> <li>None</li> </ul>
Notes	<ul style="list-style-type: none"> <li>The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled using <code>__HAL_WWDG_ENABLE_IT()</code> macro. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.</li> </ul>

### 63.2.15 HAL\_WWDG\_WakeupCallback

Function Name	<b>void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwwdg)</b>
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</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>

### 63.2.16 HAL\_WWDG\_GetState

Function Name	<b>HAL_WWDG_StateTypeDef HAL_WWDG_GetState (WWDG_HandleTypeDef * hwwdg)</b>
---------------	---



Function Description	Returns the WWDG state.
Parameters	<ul style="list-style-type: none"> <li><b>hwwdg</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>

## 63.3 WWDG Firmware driver defines

### 63.3.1 WWDG

#### *WWDG Exported Macros*

<code>__HAL_WWDG_RESET_HANDLE_STATE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Reset WWDG handle state.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: WWDG handle</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None</li> </ul>
<code>__HAL_WWDG_ENABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Enables the WWDG peripheral.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: WWDG handle</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None</li> </ul>
<code>__HAL_WWDG_DISABLE</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Disables the WWDG peripheral.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: WWDG handle</li> </ul> <p><b>Return value:</b></p> <ul style="list-style-type: none"> <li>None</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>WARNING: This is a dummy macro for HAL code alignment. Once enable, WWDG Peripheral cannot be disabled except by a system reset.</li> </ul>
<code>__HAL_WWDG_GET_IT</code>	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>Gets the selected WWDG's it status.</li> </ul> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li><code>__HANDLE__</code>: WWDG handle</li> <li><code>__INTERRUPT__</code>: specifies the it to check. This parameter can be one of the following values:</li> </ul>

- WWDG\_FLAG\_EWIF: Early wakeup interrupt IT

**Return value:**

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

**Description:**

- Clear the WWDG's interrupt pending bits bits to clear the selected interrupt pending bits.

**Parameters:**

- \_\_HANDLE\_\_: WWDG handle
- \_\_INTERRUPT\_\_: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG\_FLAG\_EWIF: Early wakeup interrupt flag

**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.

**Description:**

- Disables the WWDG early wakeup interrupt.

**Parameters:**

- \_\_HANDLE\_\_: WWDG handle
- \_\_INTERRUPT\_\_: specifies the interrupt to disable. This parameter can be one of the following values:
  - WWDG\_IT\_EWI: Early wakeup interrupt

**Return value:**

- None

\_\_HAL\_WWDG\_CLEAR\_IT

\_\_HAL\_WWDG\_ENABLE\_IT

\_\_HAL\_WWDG\_DISABLE\_IT

`__HAL_WWDG_GET_FLAG`**Notes:**

- **WARNING:** This is a dummy macro for HAL code alignment. Once enabled this interrupt cannot be disabled except by a system reset.

**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

## 64 FAQs

### General subjects

#### Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not required in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, ...). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL\_UART\_Init() then HAL\_UART\_Transmit() or HAL\_UART\_Receive().

#### Which STM32F7 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32F7 devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to [Section 4.4: "Devices supported by HAL drivers"](#).

#### What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

### Architecture

#### How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32f7xx\_hal\_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration...)

A template is provided in the HAL drivers folders (stm32f7xx\_hal\_conf\_template.c).

### Which header files should I include in my application to use the HAL drivers?

Only stm32f7xx\_hal.h file has to be included.

### What is the difference between stm32f7xx\_hal\_ppp.c/h and stm32f7xx\_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 (stm32f7xx\_hal\_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32f7xx\_hal\_ppp\_ex.c): It includes the specific APIs for specific device part number or family.

### Initialization and I/O operation functions

#### How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system\_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 functions 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 SysTick 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.



## 65 Revision history

Table 18: Document revision history

Date	Revision	Changes
22-Jun-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