

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

1	Acronyms and definitions.....	54
2	Overview of HAL drivers	56
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

3 HAL System Driver	90
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_DelInit	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
4 HAL ADC Generic Driver.....	98
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_DelInit.....	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
5	HAL ADC Extension Driver	118
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
6	HAL CAN Generic Driver.....	127
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_Delinit.....	133
6.2.8	HAL_CAN_MspInit	133
6.2.9	HAL_CAN_MspDelinit.....	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
7	HAL CEC Generic Driver	144
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_DelInit.....	147
7.2.7	HAL_CEC_MspInit	147
7.2.8	HAL_CEC_MspDelInit.....	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
8	HAL CORTEX Generic Driver.....	160
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
9	HAL CRC Generic Driver.....	170
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

10 HAL CRC Extension Driver	177
10.1 CRCEEx Firmware driver API description	177
10.1.1 CRC Extended features functions	177
10.1.2 HAL_CRCEEx_Polynomial_Set	177
10.1.3 HAL_CRCEEx_Input_Data_Reverse	177
10.1.4 HAL_CRCEEx_Output_Data_Reverse.....	178
10.2 CRCEEx Firmware driver defines.....	178
10.2.1 CRCEEx.....	178
11 HAL CRYP Generic Driver.....	180
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_Delinit	185
11.2.11 HAL_CRYP_MspInit	185
11.2.12 HAL_CRYP_MspDelinit	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
12	HAL CRYP Extension Driver.....	203
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
13	HAL DAC Generic Driver	210
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_DelInit.....	214
13.2.9	HAL_DAC_MspInit	214
13.2.10	HAL_DAC_MspDelInit.....	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
14	HAL DAC Extension Driver	224
14.1	DACEEx 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_DACEEx_DualGetValue	224
14.1.4	HAL_DACEEx_TriangleWaveGenerate	224
14.1.5	HAL_DACEEx_NoiseWaveGenerate	225
14.1.6	HAL_DACEEx_DualSetValue.....	226
14.1.7	HAL_DACEEx_ConvCpltCallbackCh2	226
14.1.8	HAL_DACEEx_ConvHalfCpltCallbackCh2	226
14.1.9	HAL_DACEEx_ErrorCallbackCh2	227
14.1.10	HAL_DACEEx_DMAUnderrunCallbackCh2	227
14.2	DACEEx Firmware driver defines	227
14.2.1	DACEEx	227
15	HAL DCMI Generic Driver	229
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_DelInit	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
16	HAL DCMI Extension Driver.....	240
16.1	DCMIEF Firmware driver registers structures.....	240
16.1.1	DCMI_CodesInitTypeDef.....	240
16.1.2	DCMI_InitTypeDef	240
16.2	DCMIEF 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	DCMIEF Firmware driver defines	242
16.3.1	DCMIEF	242
17	HAL DMA2D Generic Driver.....	243
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_DelInit.....	248
17.2.8	HAL_DMA2D_MspInit	248
17.2.9	HAL_DMA2D_MspDelInit.....	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
18	HAL DMA Generic Driver	259
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_Delnit	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
19	HAL DMA Extension Driver.....	269
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
20	HAL ETH Generic Driver	271
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_DMARxFrameInfos.....	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_Delnit	279
20.2.8	HAL_ETH_DMATxDescListInit.....	279
20.2.9	HAL_ETH_DMARxDescListInit	280
20.2.10	HAL_ETH_MspInit.....	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
21	HAL FLASH Generic Driver.....	315
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_OperationErrorHandler	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

Contents	UM1905
21.3 FLASH Firmware driver defines	319
21.3.1 FLASH	319
22 HAL FLASH Extension Driver	324
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
23 HAL GPIO Generic Driver.....	330
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_DelInit	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
24 HAL GPIO Extension Driver	339
24.1 GPIOEx Firmware driver defines.....	339
24.1.1 GPIOEx	339

25 HAL HASH Generic Driver	340
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_DelInit	351
25.2.36	HAL_HASH_MspInit	351
25.2.37	HAL_HASH_MspDelInit	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_MspDelInit	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
26	HAL HASH Extension Driver.....	357
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
27	HAL HCD Generic Driver.....	369
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_DelInit	371
27.2.10	HAL_HCD_MspInit	371
27.2.11	HAL_HCD_MspDelInit.....	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
28	HAL I2C Generic Driver	377
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
29	HAL I2C Extension Driver	398
29.1	I2CEEx 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_I2CEEx_ConfigAnalogFilter	398
29.1.5	HAL_I2CEEx_ConfigDigitalFilter	398
29.2	I2CEEx Firmware driver defines	399
29.2.1	I2CEEx	399
30	HAL I2S Generic Driver	400
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
31	HAL IRDA Generic Driver	414
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_DMAPause.....	421
31.2.16	HAL_IRDA_DMAResume.....	421
31.2.17	HAL_IRDA_DMAShort.....	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
32	HAL IRDA Extension Driver	431
32.1	IRDAEx Firmware driver defines	431
32.1.1	IRDAEx	431
33	HAL IWDG Generic Driver.....	432
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_MspInit	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
34	HAL LPTIM Generic Driver.....	438
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_DelInit.....	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
35	HAL LTDC Generic Driver	458
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
36	HAL NAND Generic Driver	475
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_DelInit	478
36.2.8	HAL_NAND_MspInit.....	478
36.2.9	HAL_NAND_MspDelInit	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
37	HAL NOR Generic Driver.....	485
37.1	NOR Firmware driver registers structures	485
37.1.1	NOR_IDTypeDef	485
37.1.2	NOR_CFITypeDef	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_DelInit	488
37.2.8	HAL_NOR_MspInit.....	488
37.2.9	HAL_NOR_MspDelInit	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
38	HAL PCD Generic Driver	494
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_DelInit.....	496
38.2.8	HAL_PCD_MsplInit	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
39	HAL PCD Extension Driver	505
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
40	HAL PWR Generic Driver	507
40.1	PWR Firmware driver registers structures	507
40.1.1	PWR_PVDTTypeDef	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_DelInit.....	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
41	HAL PWR Extension Driver	519
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
42	HAL QSPI Generic Driver	527
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_DelInit.....	533
42.2.7	HAL_QSPI_MspInit	533
42.2.8	HAL_QSPI_MspDelInit.....	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
43	HAL RCC Generic Driver	549
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_Delinit	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
44	HAL RCC Extension Driver	580
44.1	RCCEEx Firmware driver registers structures	580
44.1.1	RCC_PLLI2SInitTypeDef	580
44.1.2	RCC_PLLSAInitTypeDef	580
44.1.3	RCC_PерiphCLKInitTypeDef	581
44.2	RCCEEx Firmware driver API description	583
44.2.1	Extended Peripheral Control functions	583
44.2.2	HAL_RCCEEx_PерiphCLKConfig	583
44.2.3	HAL_RCCEEx_GetPeriphCLKConfig	584

44.2.4	HAL_RCCEx_GetPeriphCLKFreq	584
44.3	RCCEx Firmware driver defines	584
44.3.1	RCCEx	584
45	HAL RNG Generic Driver	622
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_Delnit	623
45.2.7	HAL_RNG_MspInit	623
45.2.8	HAL_RNG_MspDelnit	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
46	HAL RTC Generic Driver	629
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_DelInit.....	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_DelInit.....	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
47	HAL RTC Extension Driver	654
47.1	RTCEEx Firmware driver registers structures	654
47.1.1	RTC_TamperTypeDef	654
47.2	RTCEEx 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_RTCEEx_SetTimeStamp	657
47.2.7	HAL_RTCEEx_SetTimeStamp_IT	657
47.2.8	HAL_RTCEEx_DeactivateTimeStamp	658
47.2.9	HAL_RTCEEx_SetInternalTimeStamp	658
47.2.10	HAL_RTCEEx_DeactivateInternalTimeStamp	658
47.2.11	HAL_RTCEEx_GetTimeStamp.....	659
47.2.12	HAL_RTCEEx_SetTamper	659
47.2.13	HAL_RTCEEx_SetTamper_IT	659
47.2.14	HAL_RTCEEx_DeactivateTamper	659
47.2.15	HAL_RTCEEx_TamperTimeStampIRQHandler	660
47.2.16	HAL_RTCEEx_TimeStampEventCallback	660
47.2.17	HAL_RTCEEx_Tamper1EventCallback	660
47.2.18	HAL_RTCEEx_Tamper2EventCallback	660
47.2.19	HAL_RTCEEx_Tamper3EventCallback	660
47.2.20	HAL_RTCEEx_PollForTimeStampEvent.....	661
47.2.21	HAL_RTCEEx_PollForTamper1Event.....	661
47.2.22	HAL_RTCEEx_PollForTamper2Event	661
47.2.23	HAL_RTCEEx_PollForTamper3Event	661
47.2.24	HAL_RTCEEx_SetWakeUpTimer	661
47.2.25	HAL_RTCEEx_SetWakeUpTimer_IT	662
47.2.26	HAL_RTCEEx_DeactivateWakeUpTimer.....	662
47.2.27	HAL_RTCEEx_GetWakeUpTimer	662
47.2.28	HAL_RTCEEx_WakeUpTimerIRQHandler	662
47.2.29	HAL_RTCEEx_WakeUpTimerEventCallback	663
47.2.30	HAL_RTCEEx_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_RTCEEx_SetSmoothCalib	673
47.2.71	HAL_RTCEEx_SetSynchroShift	673
47.2.72	HAL_RTCEEx_SetCalibrationOutPut	674
47.2.73	HAL_RTCEEx_DeactivateCalibrationOutPut	674
47.2.74	HAL_RTCEEx_SetRefClock	674
47.2.75	HAL_RTCEEx_DeactivateRefClock	675
47.2.76	HAL_RTCEEx_EnableBypassShadow	675
47.2.77	HAL_RTCEEx_DisableBypassShadow	675
47.2.78	HAL_RTCEEx_AlarmBEventCallback	675
47.2.79	HAL_RTCEEx_PollForAlarmBEvent	675
47.3	RTCEEx Firmware driver defines	676
47.3.1	RTCEx	676
48	HAL SAI Generic Driver	697
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_DelInit	704
48.2.8	HAL_SAI_MspInit	705
48.2.9	HAL_SAI_MspDelInit	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
49	HAL SAI Extension Driver.....	718
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
50	HAL SDRAM Generic Driver	719
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_DelInit	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
51	HAL SD Generic Driver	727
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_DelInit	735
51.2.8	HAL_SD_MspInit	735
51.2.9	HAL_SD_MspDelInit	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
52	HAL SMARTCARD Generic Driver.....	753
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_DelInit	758
52.2.7	HAL_SMARTCARD_MspInit	758
52.2.8	HAL_SMARTCARD_MspDelInit	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
53	HAL SMARTCARD Extension Driver.....	771
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
54	HAL SPDIFRX Generic Driver	773
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_DMAStop	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
55	HAL SPI Generic Driver.....	788
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_DMADeInit.....	795
55.2.19	HAL_SPI_DMAPause	795
55.2.20	HAL_SPI_DMAStop	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
56	HAL SRAM Generic Driver	805
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_DelInit	807
56.2.8	HAL_SRAM_MspInit	807
56.2.9	HAL_SRAM_MspDelInit	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
57	HAL TIM Generic Driver	812
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_DelInit.....	825
57.2.25	HAL_TIM_OC_MspInit	825
57.2.26	HAL_TIM_OC_MspDelInit.....	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_DelInit	827
57.2.35	HAL_TIM_PWM_MspInit.....	827
57.2.36	HAL_TIM_PWM_MspDelInit	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_DelInit	830
57.2.45	HAL_TIM_IC_MspInit	830
57.2.46	HAL_TIM_IC_MspDelInit	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_DelInit	832
57.2.55	HAL_TIM_OnePulse_MspInit	832
57.2.56	HAL_TIM_OnePulse_MspDelInit	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_MsplInit	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

58 HAL TIM Extension Driver.....	858
58.1 TIMEEx 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 TIMEEx 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_TIMEEx_HallSensor_Init.....	863
58.2.11 HAL_TIMEEx_HallSensor_DelInit	863
58.2.12 HAL_TIMEEx_HallSensor_MspInit.....	863
58.2.13 HAL_TIMEEx_HallSensor_MspDelInit	863
58.2.14 HAL_TIMEEx_HallSensor_Start.....	864
58.2.15 HAL_TIMEEx_HallSensor_Stop	864
58.2.16 HAL_TIMEEx_HallSensor_Start_IT.....	864
58.2.17 HAL_TIMEEx_HallSensor_Stop_IT	864
58.2.18 HAL_TIMEEx_HallSensor_Start_DMA.....	864
58.2.19 HAL_TIMEEx_HallSensor_Stop_DMA	865
58.2.20 HAL_TIMEEx_OCN_Start.....	865
58.2.21 HAL_TIMEEx_OCN_Stop	865
58.2.22 HAL_TIMEEx_OCN_Start_IT	866
58.2.23 HAL_TIMEEx_OCN_Stop_IT	866
58.2.24 HAL_TIMEEx_OCN_Start_DMA	866
58.2.25 HAL_TIMEEx_OCN_Stop_DMA.....	866
58.2.26 HAL_TIMEEx_PWMN_Start	867
58.2.27 HAL_TIMEEx_PWMN_Stop	867
58.2.28 HAL_TIMEEx_PWMN_Start_IT	867
58.2.29 HAL_TIMEEx_PWMN_Stop_IT	868
58.2.30 HAL_TIMEEx_PWMN_Start_DMA	868
58.2.31 HAL_TIMEEx_PWMN_Stop_DMA	868
58.2.32 HAL_TIMEEx_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_CommuationCallback	874
58.2.47	HAL_TIMEx_BreakCallback	874
58.2.48	HAL_TIMEx_DMACommuationCplt	875
58.2.49	HAL_TIMEx_HallSensor_GetState	875
58.3	TIMEx Firmware driver defines	875
58.3.1	TIMEX	875
59	HAL UART Generic Driver.....	879
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_DMAPause	888
59.2.19	HAL_UART_DMAResume	888
59.2.20	HAL_UART_DMAStop	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
60	HAL UART Extension Driver	909
60.1	UARTEX Firmware driver defines	909
60.1.1	UARTEX	909
61	HAL USART Generic Driver	910
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_Delnit.....	915
61.2.7	HAL_USART_MsplInit.....	915
61.2.8	HAL_USART_MspDelnit	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
62	HAL USART Extension Driver	927
62.1	USARTEx Firmware driver defines	927
62.1.1	USARTEx	927
63	HAL WWDG Generic Driver	928
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: USART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

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

2.1 HAL and user-application files

2.1.1 HAL driver files

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

Table 2: HAL drivers files

File	Description
<i>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>stm32f7xxxx_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:stm32f7xxxx_hal_adc.h,stm32f7xxxx_hal_irda.h, ...</i>

File	Description
<i>stm32f7xxxx_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>stm32f7xxxx_hal_adc_ex.c,stm32f7xxxx_hal_dma_ex.c, ...</i>
<i>stm32f7xxxx_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:</i> <i>stm32f7xx_xx_hal_adc_ex.h,stm32f7xxxx_hal_dma_ex.h, ...</i>
<i>stm32f7xxxx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32f7xxxx_hal.h</i>	<i>stm32f7xxxx_hal.c</i> header file
<i>stm32f7xxxx_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>stm32f7xxxx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32f7xxxx_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">• relocate the vector table in internal SRAM.
<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 (optional)</i>	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>	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. . The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
<i>main.c/h</i>	This file contains the main program routine, mainly: <ul style="list-style-type: none"> • the call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

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

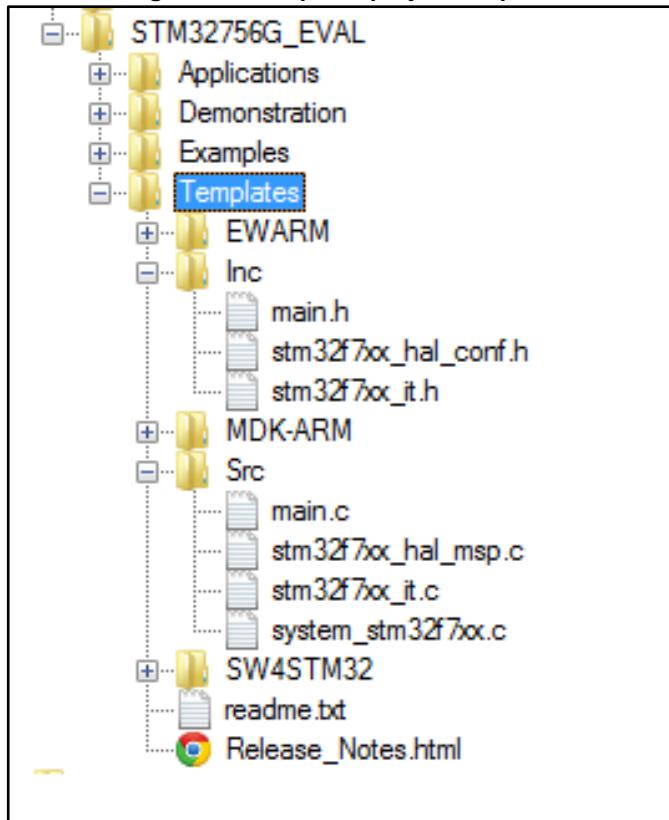
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- It contains sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows to configure the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
 - HAL is initialized
 - SysTick ISR implemented for HAL_Delay()
 - System clock configured with the maximum frequency of the device



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

Figure 1: Example of project template



2.2 HAL data structures

Each HAL driver can contain the following data structures:

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

2.2.1 Peripheral handle structures

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

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

The peripheral handle is used for the following purposes:

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

An example of peripheral structure is shown below:

```

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

```



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

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



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



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

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

2.2.2 Initialization and configuration structure

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

```

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

```

```

in a frame.*/
uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
uint32_t Parity; /*!< Specifies the parity mode. */
uint32_t Mode; /*!< Specifies 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
Common APIs	X	X ⁽¹⁾
Family specific APIs		X
Device specific APIs		X

Notes:

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



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



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

2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

IP/Module	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
File names	<i>stm32f7xx_hal_ppp (c/h)</i>	<i>stm32f7xx_hal_ppp_ex (c/h)</i>	<i>stm32f7xx_hal_ppp_ex (c/h)</i>
Module name	<i>HAL_PPP_MODULE</i>		
Function name	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>	<i>HAL_PPPEx_Function</i> <i>HAL_PPPEx_FeatureFunction_MODE</i>
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	NA	NA

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the *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_Delnit*, e.g. *TIM_Delnit*.
- 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 fromstm32f7xx_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_MspDlInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8: Callback functions

Callback functions	Example
HAL_PPP_MspInit() / _DlInit()	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_DlInit()
- IO operation functions:** HAL_PPP_Read(), HAL_PPP_Write(), HAL_PPP_Transmit(), HAL_PPP_Receive()
- Control functions:** HAL_PPP_Set (), HAL_PPP_Get () .
- State and Errors functions:** HAL_PPP_GetState (), HAL_PPP_GetError () .

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

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

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

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

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

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

Table 9: HAL generic APIs

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

2.7 HAL extension APIs

2.7.1 HAL extension model overview

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

The extension model consists of an additional file, `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
<code>HAL_ADCEx_CalibrationStart()</code>	This function is used to start the automatic ADC calibration

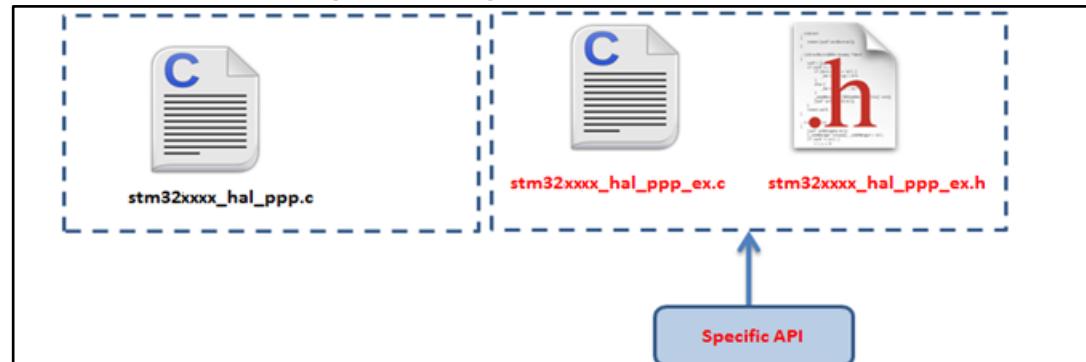
2.7.2 HAL extension model cases

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



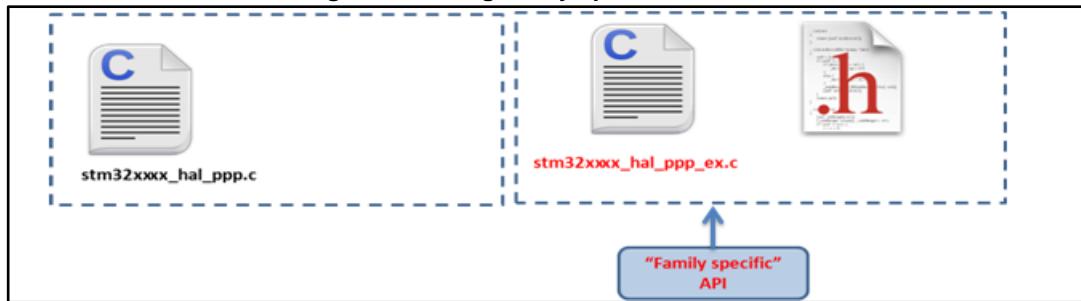
Example: `stm32f7xx_hal_hash_ex.h`

```
#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

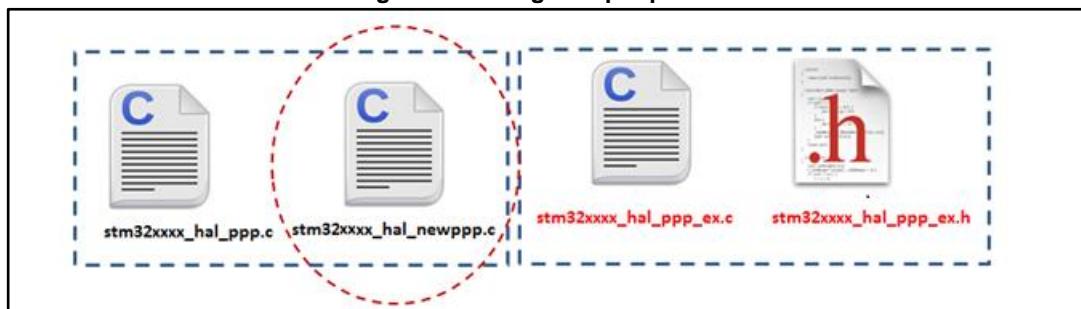


Case3 : Adding a new peripheral (specific to a device belonging to a given family)

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

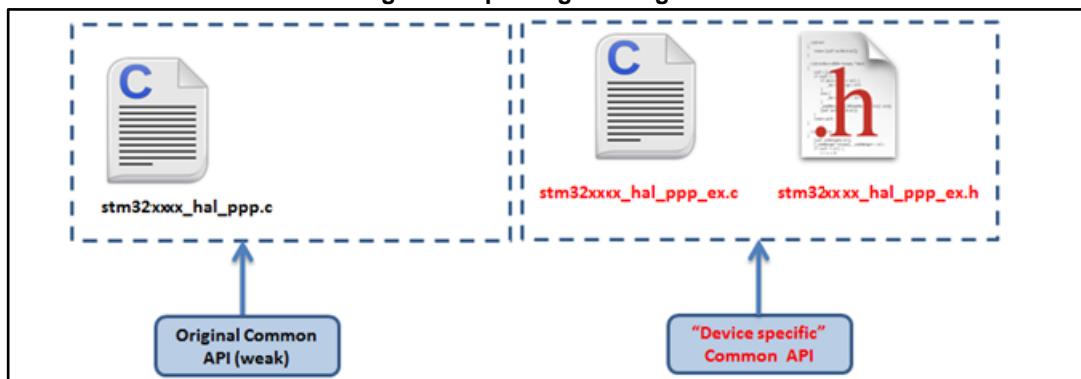


Example: xx_hal_sai.c/h

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:

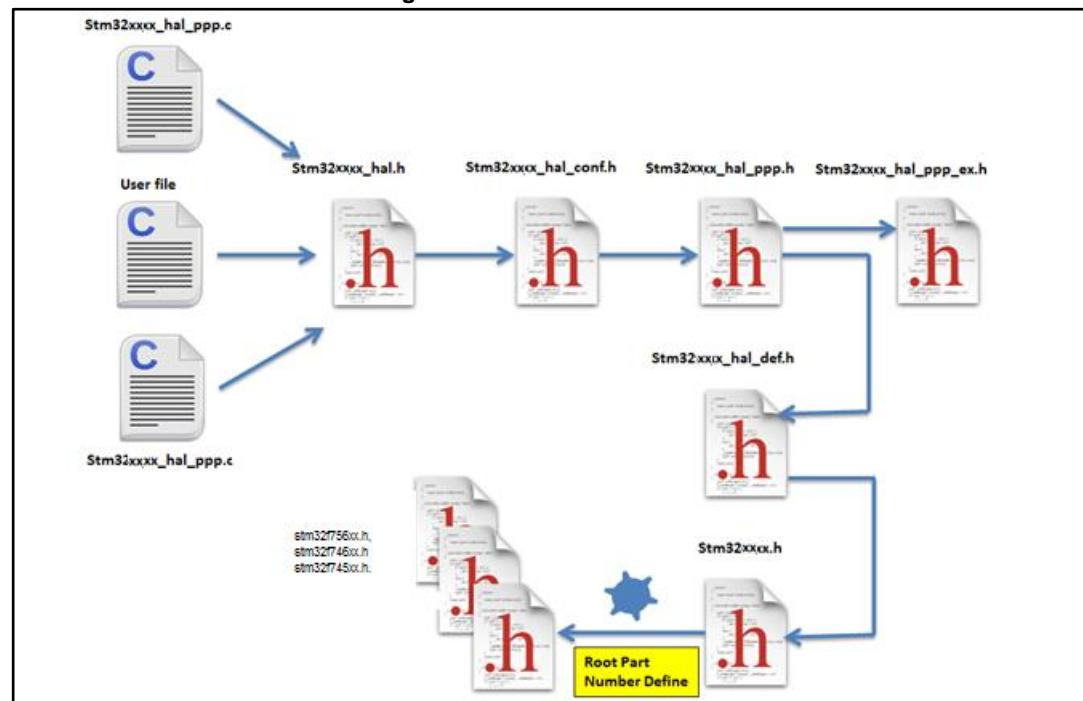
```
#if defined (STM32F756xx)
typedef struct
{
(...)

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

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



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

A set of macros are defined in `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"> • <u>GPIO mode</u> <ul style="list-style-type: none"> – <code>GPIO_MODE_INPUT</code> : Input Floating – <code>GPIO_MODE_OUTPUT_PP</code> : Output Push Pull – <code>GPIO_MODE_OUTPUT_OD</code> : Output Open Drain – <code>GPIO_MODE_AF_PP</code> : Alternate Function Push Pull – <code>GPIO_MODE_AF_OD</code> : Alternate Function Open Drain – <code>GPIO_MODE_ANALOG</code> : Analog mode • <u>External Interrupt Mode</u> <ul style="list-style-type: none"> – <code>GPIO_MODE_IT_RISING</code> : Rising edge trigger detection – <code>GPIO_MODE_IT_FALLING</code> : Falling edge trigger detection – <code>GPIO_MODE_IT_RISING_FALLING</code> : Rising/Falling edge trigger detection • <u>External Event Mode</u> <ul style="list-style-type: none"> – <code>GPIO_MODE_EVT_RISING</code> : Rising edge trigger detection – <code>GPIO_MODE_EVT_FALLING</code> : Falling edge trigger detection – <code>GPIO_MODE_EVT_RISING_FALLING</code> : Rising/Falling edge trigger detection

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_{SUBLOCK}_EXTI_ENABLE_IT()</code>	Enables a given EXTI line interrupt Example: <code>_HAL_PWR_PVD_EXTI_ENABLE_IT()</code>
<code>_HAL_PPP_{SUBLOCK}_EXTI_DISABLE_IT()</code>	Disables a given EXTI line. Example: <code>_HAL_PWR_PVD_EXTI_DISABLE_IT()</code>
<code>_HAL_PPP_{SUBLOCK}_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_{SUBLOCK}_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_</code> <code>PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()</code>	Configure an EXTI Interrupt or Event on rising edge
<code>__HAL_</code> <code>PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()</code>	Enable an EXTI Interrupt or Event on Falling edge
<code>__HAL_</code> <code>PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()</code>	Disable an EXTI Interrupt or Event on rising edge
<code>__HAL_</code> <code>PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()</code>	Disable an EXTI Interrupt or Event on Falling edge
<code>__HAL_</code> <code>PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()</code>	Enable an EXTI Interrupt or Event on Rising/Falling edge
<code>__HAL_</code> <code>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: enables 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”).



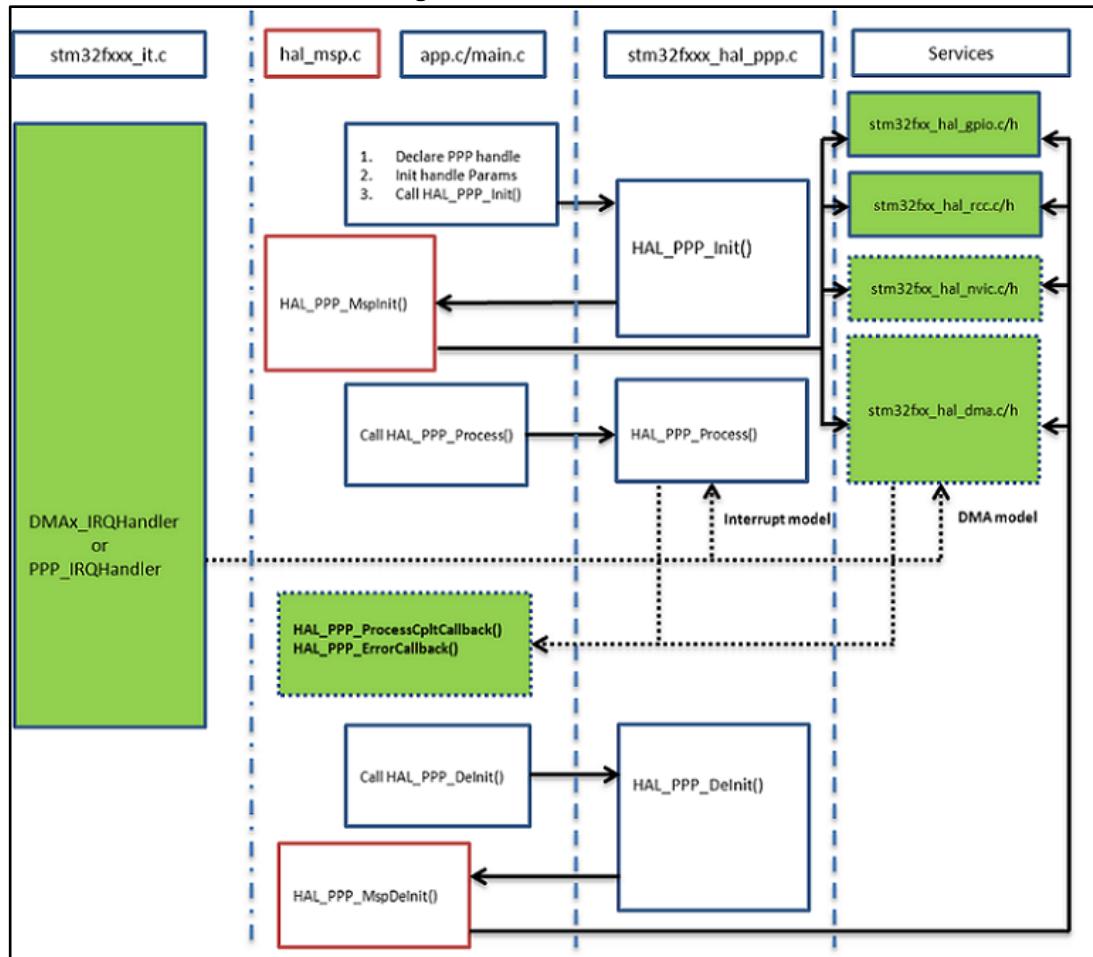
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

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_DelInit()
 - Resets all peripherals
 - Calls function HAL_MspDelInit() which a is user callback function to do system level De-Initializations.
- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.
Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

2.12.2.2 System clock initialization

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

```
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_OscInitStruct.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_HandleTypeDefDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspInit could be implemented in the user file */
}

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

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

The HAL MSP callback is implemented inside the *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
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

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

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

peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit()*.

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

2.12.3 HAL IO operation process

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

- Polling mode
- Interrupt mode
- DMA mode

2.12.3.1 Polling mode

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

The example below shows the typical polling mode processing sequence :

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

2.12.3.2 Interrupt mode

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

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

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

To use a process in interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and *HAL_PPP_IRQHandler* in *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 = USART1;
HAL_UART_Init(&UartHandle);
(...)

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

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

An example of use is illustrated below:

main.c file:

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

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) ⁽¹⁾	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

Notes:

⁽¹⁾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 */
PPP->ErrorCode = HAL_ERRORCODE; /* Set the error code */
HAL_UNLOCK(PPP); /* Unlock the PPP resources */
return HAL_ERROR; /*return with HAL error */
```

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

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

2.12.4.3 Run-time checking

The HAL implements run-time failure detection by checking the input values of all HAL drivers functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL drivers' functions which have an input parameter. It allows verifying that the input value lies within the parameter allowed values.

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

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

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

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



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

3 HAL System Driver

3.1 HAL Firmware driver API description

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	HAL_StatusTypeDef HAL_Init (void)
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"> • HAL status
Notes	<ul style="list-style-type: none"> • 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.

3.1.5 HAL_DeInit

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

3.1.6 HAL_MspInit

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

3.1.7 HAL_MspDeInit

Function Name	void HAL_MspDeInit (void)
Function Description	DeInitializes the MSP.
Return values	<ul style="list-style-type: none">None

3.1.8 HAL_InitTick

Function Name	HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)
Function Description	This function configures the source of the time base.
Parameters	<ul style="list-style-type: none">TickPriority: Tick interrupt priority.
Return values	<ul style="list-style-type: none">HAL status
Notes	<ul style="list-style-type: none">This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig().In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.The function is declared as __weak to be overwritten in case of other implementation in user file.

3.1.9 HAL_IncTick

Function Name	void HAL_IncTick (void)
Function Description	This function is called to increment a global variable "uwTick" used as application time base.
Return values	<ul style="list-style-type: none">None
Notes	<ul style="list-style-type: none">In the default implementation, this variable is incremented each 1ms in Systick ISR.This function is declared as __weak to be overwritten in case of other implementations in user file.

3.1.10 HAL_GetTick

Function Name	uint32_t HAL_GetTick (void)
Function Description	Provides a tick value in millisecond.
Return values	<ul style="list-style-type: none">tick value
Notes	<ul style="list-style-type: none">This function is declared as __weak to be overwritten in case of other implementations in user file.

3.1.11 HAL_Delay

Function Name	void HAL_Delay (__IO uint32_t Delay)
Function Description	This function provides accurate delay (in milliseconds) based on

variable incremented.

- | | |
|---------------|--|
| Parameters | <ul style="list-style-type: none"> • Delay: specifies the delay time length, in milliseconds. |
| Return values | <ul style="list-style-type: none"> • None |
| Notes | <ul style="list-style-type: none"> • 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. • This function is declared as __weak to be overwritten in case of other implementations in user file. |

3.1.12 HAL_SuspendTick

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

3.1.13 HAL_ResumeTick

- | | |
|----------------------|---|
| Function Name | void HAL_ResumeTick (void) |
| Function Description | Resume Tick increment. |
| Return values | <ul style="list-style-type: none"> • None |
| Notes | <ul style="list-style-type: none"> • 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. • This function is declared as __weak to be overwritten in case of other implementations in user file. |

3.1.14 HAL_GetHalVersion

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

3.1.15 HAL_GetREVID

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

3.1.16 HAL_GetDEVID

Function Name	uint32_t HAL_GetDEVID (void)
Function Description	Returns the device identifier.
Return values	<ul style="list-style-type: none">• Device identifier

3.1.17 HAL_DBGMCU_EnableDBGSleepMode

Function Name	void HAL_DBGMCU_EnableDBGSleepMode (void)
Function Description	Enable the Debug Module during SLEEP mode.
Return values	<ul style="list-style-type: none">• None

3.1.18 HAL_DBGMCU_DisableDBGSleepMode

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

3.1.19 HAL_DBGMCU_EnableDBGStopMode

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

3.1.20 HAL_DBGMCU_DisableDBGStopMode

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

3.1.21 HAL_DBGMCU_EnableDBGStandbyMode

Function Name	void HAL_DBGMCU_EnableDBGStandbyMode (void)
Function Description	Enable the Debug Module during STANDBY mode.
Return values	<ul style="list-style-type: none">• None

3.1.22 HAL_DBGMCU_DisableDBGStandbyMode

Function Name	void HAL_DBGMCU_DisableDBGStandbyMode (void)
Function Description	Disable the Debug Module during STANDBY mode.
Return values	<ul style="list-style-type: none">• None

3.1.23 HAL_EnableCompensationCell

Function Name	void HAL_EnableCompensationCell (void)
Function Description	Enables the I/O Compensation Cell.
Return values	<ul style="list-style-type: none">• None

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

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

3.1.26 HAL_DisableFMCMemorySwapping	
Function Name	void HAL_DisableFMCMemorySwapping (void)
Function Description	Disables the FMC Memory Mapping Swapping.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> SDRAM is accessible at 0xC0000000 (default mapping) and NOR/RAM is accessible at 0x60000000 (default mapping)

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 Low 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: $V_{REF(minus)} = V_{IN} = V_{REF(plus)}$.
14. DMA request generation during regular channel conversion.

4.2.2 How to use this driver

1. Initialize the ADC low level resources by implementing the HAL_ADC_MspInit():
 - a. Enable the ADC interface clock using `__HAL_RCC_ADC_CLK_ENABLE()`
 - b. ADC pins configuration
 - Enable the clock for the ADC GPIOs using the following function:
`__HAL_RCC_GPIOx_CLK_ENABLE()`
 - Configure these ADC pins in analog mode using `HAL_GPIO_Init()`
 - c. In case of using interrupts (e.g. `HAL_ADC_Start_IT()`)
 - Configure the ADC interrupt priority using `HAL_NVIC_SetPriority()`
 - Enable the ADC IRQ handler using `HAL_NVIC_EnableIRQ()`
 - In ADC IRQ handler, call `HAL_ADC_IRQHandler()`
 - d. In case of using DMA to control data transfer (e.g. `HAL_ADC_Start_DMA()`)
 - Enable the DMAx interface clock using
`__HAL_RCC_DMAx_CLK_ENABLE()`
 - Configure and enable two DMA streams stream for managing data transfer from peripheral to memory (output stream)
 - Associate the initialized DMA handle to the CRYP DMA handle using
`__HAL_LINKDMA()`
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream.

Configuration of ADC, groups regular/injected, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function `HAL_ADC_Init()`.

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

Execution of ADC conversions

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

Polling mode IO operation

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

Interrupt mode IO operation

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

DMA mode IO operation

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

ADC HAL driver macros list

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

- __HAL_ADC_ENABLE : Enable the ADC peripheral
- __HAL_ADC_DISABLE : Disable the ADC peripheral

- `_HAL_ADC_ENABLE_IT`: Enable the ADC end of conversion interrupt
- `_HAL_ADC_DISABLE_IT`: Disable the ADC end of conversion interrupt
- `_HAL_ADC_GET_IT_SOURCE`: Check if the specified ADC interrupt source is enabled or disabled
- `_HAL_ADC_CLEAR_FLAG`: Clear the ADC's pending flags
- `_HAL_ADC_GET_FLAG`: Get the selected ADC's flag status
- `ADC_GET_RESOLUTION`: Return resolution bits in CR1 register



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

Deinitialization of ADC

1. Disable the ADC interface
 - ADC clock can be hard reset and disabled at RCC top level.
 - Hard reset of ADC peripherals using macro `_HAL_RCC_ADC_FORCE_RESET()`, `_HAL_RCC_ADC_RELEASE_RESET()`.
 - ADC clock disable using the equivalent macro/functions as configuration step.
 - Example: Into `HAL_ADC_MspDelInit()` (recommended code location) or with other device clock parameters configuration:
 - `HAL_RCC_GetOscConfig(&RCC_OsclInitStructure);`
 - `RCC_OsclInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI;`
 - `RCC_OsclInitStructure.HSISState = RCC_HSI_OFF;` (if not used for system clock)
 - `HAL_RCC_OscConfig(&RCC_OsclInitStructure);`
2. ADC pins configuration
 - Disable the clock for the ADC GPIOs using macro `_HAL_RCC_GPIOx_CLK_DISABLE()`
3. Optionally, in case of usage of ADC with interruptions:
 - Disable the NVIC for ADC using function `HAL_NVIC_DisableIRQ(ADCx_IRQn)`
4. Optionally, in case of usage of DMA:
 - Deinitialize the DMA using function `HAL_DMA_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_DelInit\(\)`](#)
- [`HAL_ADC_MspInit\(\)`](#)
- [`HAL_ADC_MspDelInit\(\)`](#)

4.2.4 IO operation functions

This section provides functions allowing to:

- Start conversion of regular channel.
- Stop conversion of regular channel.
- Start conversion of regular channel and enable interrupt.
- Stop conversion of regular channel and disable interrupt.
- Start conversion of regular channel and enable DMA transfer.
- Stop conversion of regular channel and disable DMA transfer.
- Handle ADC interrupt request.

This section contains the following APIs:

- [*HAL_ADC_Start\(\)*](#)
- [*HAL_ADC_Stop\(\)*](#)
- [*HAL_ADC_PollForConversion\(\)*](#)
- [*HAL_ADC_PollForEvent\(\)*](#)
- [*HAL_ADC_Start_IT\(\)*](#)
- [*HAL_ADC_Stop_IT\(\)*](#)
- [*HAL_ADC_IRQHandler\(\)*](#)
- [*HAL_ADC_Start_DMA\(\)*](#)
- [*HAL_ADC_Stop_DMA\(\)*](#)
- [*HAL_ADC_GetValue\(\)*](#)
- [*HAL_ADC_ConvCpltCallback\(\)*](#)
- [*HAL_ADC_ConvHalfCpltCallback\(\)*](#)
- [*HAL_ADC_LevelOutOfWindowCallback\(\)*](#)
- [*HAL_ADC_ErrorCallback\(\)*](#)

4.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure regular channels.
- Configure injected channels.
- Configure multimode.
- Configure the analog watch dog.

This section contains the following APIs:

- [*HAL_ADC_ConfigChannel\(\)*](#)
- [*HAL_ADC_AnalogWDGConfig\(\)*](#)

4.2.6 Peripheral State and errors functions

This subsection provides functions allowing to

- Check the ADC state
- Check the ADC Error

This section contains the following APIs:

- [*HAL_ADC_GetState\(\)*](#)
- [*HAL_ADC_GetError\(\)*](#)

4.2.7 HAL_ADC_Init

Function Name	<code>HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef *hadc)</code>
Function Description	Initializes the ADCx peripheral according to the specified parameters in the ADC_InitStruct and initializes the ADC MSP.

Parameters	<ul style="list-style-type: none"> hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> 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).

4.2.8 HAL_ADC_DeInit

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

4.2.9 HAL_ADC_MspInit

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

4.2.10 HAL_ADC_MspDeInit

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

4.2.11 HAL_ADC_Start

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

4.2.12 HAL_ADC_Stop



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

4.2.13 HAL_ADC_PollForConversion

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

4.2.14 HAL_ADC_PollForEvent

Function Name	HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)
Function Description	Poll for conversion event.
Parameters	<ul style="list-style-type: none"> hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. EventType: 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. Timeout: Timeout value in millisecond.
Return values	<ul style="list-style-type: none"> HAL status

4.2.15 HAL_ADC_Start_IT

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

4.2.16 HAL_ADC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)
Function Description	Disables the interrupt and stop ADC conversion of regular

channels.

Parameters	<ul style="list-style-type: none"> hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> HAL status.
Notes	<ul style="list-style-type: none"> Caution: This function will stop also injected channels.

4.2.17 HAL_ADC_IRQHandler

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

4.2.18 HAL_ADC_Start_DMA

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

4.2.19 HAL_ADC_Stop_DMA

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

4.2.20 HAL_ADC_GetValue

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

4.2.21 HAL_ADC_ConvCpltCallback

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

4.2.22 HAL_ADC_ConvHalfCpltCallback

Function Name	void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)
Function Description	Regular conversion half DMA transfer callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None

4.2.23 HAL_ADC_LevelOutOfWindowCallback

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

4.2.24 HAL_ADC_ErrorCallback

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

4.2.25 HAL_ADC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)
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"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • sConfig: ADC configuration structure.

Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • ADC Error Code
4.3 ADC Firmware driver defines	
4.3.1 ADC	
<i>ADC Analog Watchdog Selection</i>	
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	
<i>ADC Common Channels</i>	

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

<u>__HAL_ADC_RESET_HANDLE_STATE</u>	Description:
	<ul style="list-style-type: none">• Reset ADC handle state.
Parameters:	
	<ul style="list-style-type: none">• <u>__HANDLE__</u>: ADC handle
Return value:	
	<ul style="list-style-type: none">• None
<u>__HAL_ADC_ENABLE</u>	Description:
	<ul style="list-style-type: none">• Enable the ADC peripheral.
Parameters:	
	<ul style="list-style-type: none">• <u>__HANDLE__</u>: 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.

Return value:

- None

`__HAL_ADC_GET_FLAG`

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.

Parameters:

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

Return value:

- None

ADC_SMPR2

Description:

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

Parameters:

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

Return value:

- None

ADC_SQR3_RK

Description:

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

Parameters:

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

Return value:

- None

ADC_SQR2_RK

Description:

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

Parameters:

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

Return value:

- None

ADC_SQR1_RK

Description:

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

Parameters:

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

Return value:

- None

ADC_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

Description:

- Enable ADC scan mode.

Parameters:

- `_SCANCONV_MODE_`: Scan conversion mode.

Return value:

- None

Description:

- Enable the ADC end of conversion selection.

Parameters:

- `_EOCSelection_MODE_`: End of conversion selection mode.

Return value:

- None

Description:

- Enable the ADC DMA continuous request.

Parameters:

- `_DMAContReq_MODE_`: DMA continuous request mode.

Return value:

- None

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 = 0xFFFF.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***
Specifies the number of ADC conversions that will be done using the sequencer for injected channel group. This parameter must be a number between Min_Data = 1 and Max_Data = 4.
- ***uint32_t ADC_InjectionConfTypeDef::AutoInjectedConv***
Enables or disables the selected ADC automatic injected group conversion after regular one
- ***uint32_t ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***
Specifies whether the conversion is performed in Discontinuous mode or not for injected channels. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***
Select the external trigger edge and enable the trigger of an injected channels. This parameter can be a value of [*ADCEx_External_trigger_edge_Injected*](#)
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***
Select the external event used to trigger the start of conversion of a injected channels. This parameter can be a value of [*ADCEx_External_trigger_Source_Injected*](#)

5.1.2 ADC_MultiModeTypeDef

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

5.2.4 HAL_ADCEx_InjectedStart_IT

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

5.2.5 HAL_ADCEx_InjectedStop

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

Notes

- Caution: This function will stop also regular channels.

5.2.6 HAL_ADCEx_InjectedPollForConversion

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

5.2.7 HAL_ADCEx_InjectedStop_IT

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

Notes

- Caution: This function will stop also regular channels.

5.2.8 HAL_ADCEx_InjectedGetValue

Function Name	uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef * hadc, uint32_t InjectedRank)
Function Description	Gets the converted value from data register of injected channel.
Parameters	<ul style="list-style-type: none"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • InjectedRank: 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
Return values	<ul style="list-style-type: none"> • None

5.2.9 HAL_ADCEx_MultiModeStart_DMA

Function Name	HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)
Function Description	Enables ADC DMA request after last transfer (Multi-ADC mode) and enables ADC peripheral.
Parameters	<ul style="list-style-type: none"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • pData: Pointer to buffer in which transferred from ADC peripheral to memory will be stored. • Length: The length of data to be transferred from ADC

	peripheral to memory.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • Caution: This function must be used only with the ADC master.
5.2.10 HAL_ADCEx_MultiModeStop_DMA	
Function Name	HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA(ADC_HandleTypeDef * hadc)
Function Description	Disables ADC DMA (multi-ADC mode) and disables ADC peripheral.
Parameters	<ul style="list-style-type: none"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • HAL status
5.2.11 HAL_ADCEx_MultiModeGetValue	
Function Name	uint32_t HAL_ADCEx_MultiModeGetValue(ADC_HandleTypeDef * hadc)
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"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • The converted data value.
5.2.12 HAL_ADCEx_InjectedConvCpltCallback	
Function Name	void HAL_ADCEx_InjectedConvCpltCallback(ADC_HandleTypeDef * hadc)
Function Description	Injected conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
Return values	<ul style="list-style-type: none"> • None
5.2.13 HAL_ADCEx_InjectedConfigChannel	
Function Name	HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel(ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)
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"> • hadc: pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • sConfigInjected: ADC configuration structure for injected channel.
Return values	<ul style="list-style-type: none"> • None

5.2.14 HAL_ADCEx_MultiModeConfigChannel

Function Name	<code>HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel(ADC_HandleTypeDef * hadc, ADC_MultiModeTypeDef * multimode)</code>
Function Description	Configures the ADC multi-mode.
Parameters	<ul style="list-style-type: none"> • hadc: : pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC. • multimode: : pointer to an ADC_MultiModeTypeDef structure that contains the configuration information for multimode.
Return values	<ul style="list-style-type: none"> • HAL status

5.3 ADCEx Firmware driver defines

5.3.1 ADCEx

ADC Specific Channels

`ADC_CHANNEL_TEMPSENSOR`

ADC Common Mode

`ADC_MODE_INDEPENDENT`

`ADC_DUALMODE_REGSIMULT_INJECSIMULT`

`ADC_DUALMODE_REGSIMULT.AlterTrig`

`ADC_DUALMODE_INJECSIMULT`

`ADC_DUALMODE_REGSIMULT`

`ADC_DUALMODE_INTERL`

`ADC_DUALMODE.AlterTrig`

`ADC_TRIPLEMODE_REGSIMULT_INJECSIMULT`

`ADC_TRIPLEMODE_REGSIMULT.AlterTrig`

`ADC_TRIPLEMODE_INJECSIMULT`

`ADC_TRIPLEMODE_REGSIMULT`

`ADC_TRIPLEMODE_INTERL`

`ADC_TRIPLEMODE.AlterTrig`

ADC Direct Memory Access Mode For Multi Mode

`ADC_DMAACCESSMODE_DISABLED` DMA mode disabled

`ADC_DMAACCESSMODE_1` DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)

`ADC_DMAACCESSMODE_2` DMA mode 2 enabled (2 / 3 half-words by pairs - 2&1 then 1&3 then 3&2)

`ADC_DMAACCESSMODE_3` DMA mode 3 enabled (2 / 3 bytes by pairs - 2&1 then 1&3 then 3&2)

ADC External Trigger Edge Injected

ADC_EXTERNALTRIGINJECCONVEDGE_NONE
 ADC_EXTERNALTRIGINJECCONVEDGE_RISING
 ADC_EXTERNALTRIGINJECCONVEDGE_FALLING
 ADC_EXTERNALTRIGINJECCONVEDGE_RISINGFALLING

ADC External Trigger Source Injected

ADC_EXTERNALTRIGINJECCONV_T1_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_FilterTypeDef::FilterActivation***
Enable or disable the filter. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t CAN_FilterTypeDef::BankNumber***
Select the start slave bank filter. This parameter must be a number between Min_Data = 0 and Max_Data = 28

6.1.3 CanTxMsgTypeDef

Data Fields

- ***uint32_t StdId***
- ***uint32_t ExtId***
- ***uint32_t IDE***
- ***uint32_t RTR***
- ***uint32_t DLC***
- ***uint8_t Data***

Field Documentation

- ***uint32_t CanTxMsgTypeDef::StdId***
Specifies the standard identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x7FF
- ***uint32_t CanTxMsgTypeDef::ExtId***
Specifies the extended identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x1FFFFFFF
- ***uint32_t CanTxMsgTypeDef::IDE***
Specifies the type of identifier for the message that will be transmitted. This parameter can be a value of [CAN_Identifier_Type](#)
- ***uint32_t CanTxMsgTypeDef::RTR***
Specifies the type of frame for the message that will be transmitted. This parameter can be a value of [CAN_remote_transmission_request](#)
- ***uint32_t CanTxMsgTypeDef::DLC***
Specifies the length of the frame that will be transmitted. This parameter must be a number between Min_Data = 0 and Max_Data = 8
- ***uint8_t CanTxMsgTypeDef::Data[8]***
Contains the data to be transmitted. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF

6.1.4 CanRxMsgTypeDef

Data Fields

- ***uint32_t StdId***
- ***uint32_t ExtId***
- ***uint32_t IDE***
- ***uint32_t RTR***
- ***uint32_t DLC***
- ***uint8_t Data***
- ***uint32_t FMI***

- *uint32_t FIFONumber*

Field Documentation

- ***uint32_t CanRxMsgTypeDef::StdId***
Specifies the standard identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x7FF
- ***uint32_t CanRxMsgTypeDef::ExtId***
Specifies the extended identifier. This parameter must be a number between Min_Data = 0 and Max_Data = 0x1FFFFFFF
- ***uint32_t CanRxMsgTypeDef::IDE***
Specifies the type of identifier for the message that will be received. This parameter can be a value of [CAN_Identifier_Type](#)
- ***uint32_t CanRxMsgTypeDef::RTR***
Specifies the type of frame for the received message. This parameter can be a value of [CAN_remote_transmission_request](#)
- ***uint32_t CanRxMsgTypeDef::DLC***
Specifies the length of the frame that will be received. This parameter must be a number between Min_Data = 0 and Max_Data = 8
- ***uint8_t CanRxMsgTypeDef::Data[8]***
Contains the data to be received. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF
- ***uint32_t CanRxMsgTypeDef::FMI***
Specifies the index of the filter the message stored in the mailbox passes through. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFF
- ***uint32_t CanRxMsgTypeDef::FIFONumber***
Specifies the receive FIFO number. This parameter can be CAN_FIFO0 or CAN_FIFO1

6.1.5 CAN_HandleTypeDef

Data Fields

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

Field Documentation

- ***CAN_TypeDef* CAN_HandleTypeDef::Instance***
Register base address
- ***CAN_InitTypeDef CAN_HandleTypeDef::Init***
CAN required parameters
- ***CanTxMsgTypeDef* CAN_HandleTypeDef::pTxMsg***
Pointer to transmit structure
- ***CanRxMsgTypeDef* CAN_HandleTypeDef::pRxMsg***
Pointer to reception structure

- **`_IO HAL_CAN_StateTypeDef CAN_HandleTypeDef::State`**
CAN communication state
- **`HAL_LockTypeDef CAN_HandleTypeDef::Lock`**
CAN locking object
- **`_IO uint32_t CAN_HandleTypeDef::ErrorCode`**
CAN Error code

6.2 CAN Firmware driver API description

6.2.1 How to use this driver

1. Enable the CAN controller interface clock using `_HAL_RCC_CAN1_CLK_ENABLE()` for CAN1 and `_HAL_RCC_CAN2_CLK_ENABLE()` for CAN2 In case you are using CAN2 only, you have to enable the CAN1 clock.
2. CAN pins configuration
 - Enable the clock for the CAN GPIOs using the following function:
`_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_DelInit\(\)`](#)
- [`HAL_CAN_MspInit\(\)`](#)
- [`HAL_CAN_MspDelInit\(\)`](#)

6.2.3 IO operation functions

This section provides functions allowing to:

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

This section contains the following APIs:

- [`HAL_CAN_Transmit\(\)`](#)
- [`HAL_CAN_Transmit_IT\(\)`](#)
- [`HAL_CAN_Receive\(\)`](#)
- [`HAL_CAN_Receive_IT\(\)`](#)
- [`HAL_CAN_Sleep\(\)`](#)
- [`HAL_CAN_WakeUp\(\)`](#)
- [`HAL_CAN_IRQHandler\(\)`](#)
- [`HAL_CAN_TxCpltCallback\(\)`](#)
- [`HAL_CAN_RxCpltCallback\(\)`](#)
- [`HAL_CAN_ErrorCallback\(\)`](#)

6.2.4 Peripheral State and Error functions

This subsection provides functions allowing to :

- Check the CAN state.
- Check CAN Errors detected during interrupt process

This section contains the following APIs:

- [`HAL_CAN_GetState\(\)`](#)
- [`HAL_CAN_GetError\(\)`](#)

6.2.5 HAL_CAN_Init

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

6.2.6 HAL_CAN_ConfigFilter

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

6.2.7 HAL_CAN_DeInit

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

6.2.8 HAL_CAN_MspInit

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

6.2.9 HAL_CAN_MspDeInit

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

Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. Timeout: Specify Timeout value
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hcan: Pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN. FIFONumber: Specify the FIFO number
Return values	<ul style="list-style-type: none"> 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"> • hcan: pointer to a CAN_HandleTypeDef structure that contains the configuration information for the specified CAN.
Return values	<ul style="list-style-type: none"> • HAL status.

6.2.15 HAL_CAN_WakeUp

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

6.2.16 HAL_CAN_IRQHandler

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

6.2.17 HAL_CAN_TxCpltCallback

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

6.2.18 HAL_CAN_RxCpltCallback

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

6.2.19 HAL_CAN_ErrorCallback

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

6.2.20 HAL_CAN_GetState

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

6.2.21 HAL_CAN_GetError

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

6.3 CAN Firmware driver defines

6.3.1 CAN

CAN Exported Macros

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

- None

_HAL_CAN_MSG_PENDING**Description:**

- Return the number of pending received messages.

Parameters:

- _HANDLE_: CAN handle
- _FIFONUMBER_: Receive FIFO number, CAN_FIFO0 or CAN_FIFO1.

Return value:

- The: number of pending message.

_HAL_CAN_GET_FLAG**Description:**

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

Parameters:

- _HANDLE_: CAN Handle
- _FLAG_: specifies the flag to check. This parameter can be one of the following values:
 - CAN_TSR_RQCP0: Request MailBox0 Flag
 - CAN_TSR_RQCP1: Request MailBox1 Flag
 - CAN_TSR_RQCP2: Request MailBox2 Flag
 - CAN_FLAG_TXOK0: Transmission OK MailBox0 Flag
 - CAN_FLAG_TXOK1: Transmission OK MailBox1 Flag
 - CAN_FLAG_TXOK2: Transmission OK MailBox2 Flag
 - CAN_FLAG_TME0: Transmit mailbox 0 empty Flag
 - CAN_FLAG_TME1: Transmit mailbox 1 empty Flag
 - CAN_FLAG_TME2: Transmit mailbox 2 empty Flag
 - CAN_FLAG_FMP0: FIFO 0 Message Pending Flag
 - CAN_FLAG_FF0: FIFO 0 Full Flag
 - CAN_FLAG_FOV0: FIFO 0 Overrun Flag
 - CAN_FLAG_FMP1: FIFO 1 Message Pending Flag
 - CAN_FLAG_FF1: FIFO 1 Full Flag
 - CAN_FLAG_FOV1: FIFO 1 Overrun Flag
 - CAN_FLAG_WKU: Wake up Flag
 - CAN_FLAG_SLAK: Sleep acknowledge Flag

- CAN_FLAG_SLAKI: Sleep acknowledge Flag
- CAN_FLAG_EWG: Error Warning Flag
- CAN_FLAG_EPV: Error Passive Flag
- CAN_FLAG_BOF: Bus-Off Flag

Return value:

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

_HAL_CAN_CLEAR_FLAG**Description:**

- Clear the specified CAN pending flag.

Parameters:

- __HANDLE__: CAN Handle.
- __FLAG__: specifies the flag to check.
This parameter can be one of the following values:
 - CAN_TSR_RQCP0: Request MailBox0 Flag
 - CAN_TSR_RQCP1: Request MailBox1 Flag
 - CAN_TSR_RQCP2: Request MailBox2 Flag
 - CAN_FLAG_TXOK0: Transmission OK MailBox0 Flag
 - CAN_FLAG_TXOK1: Transmission OK MailBox1 Flag
 - CAN_FLAG_TXOK2: Transmission OK MailBox2 Flag
 - CAN_FLAG_TME0: Transmit mailbox 0 empty Flag
 - CAN_FLAG_TME1: Transmit mailbox 1 empty Flag
 - CAN_FLAG_TME2: Transmit mailbox 2 empty Flag
 - CAN_FLAG_FMP0: FIFO 0 Message Pending Flag
 - CAN_FLAG_FF0: FIFO 0 Full Flag
 - CAN_FLAG_FOV0: FIFO 0 Overrun Flag
 - CAN_FLAG_FMP1: FIFO 1 Message Pending Flag
 - CAN_FLAG_FF1: FIFO 1 Full Flag
 - CAN_FLAG_FOV1: FIFO 1 Overrun Flag
 - CAN_FLAG_WKU: Wake up Flag
 - CAN_FLAG_SLAK: Sleep acknowledge Flag
 - CAN_FLAG_SLAKI: Sleep acknowledge Flag
 - CAN_FLAG_EWG: Error Warning Flag

- CAN_FLAG_EPV: Error Passive Flag
- CAN_FLAG_BOF: Bus-Off Flag

Return value:

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

[__HAL_CAN_GET_IT_SOURCE](#)**Description:**

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

Parameters:

- __HANDLE__: CAN Handle
- __INTERRUPT__: specifies the CAN interrupt source to check. This parameter can be one of the following values:
 - CAN_IT_TME: Transmit mailbox empty interrupt enable
 - CAN_IT_FMP0: FIFO0 message pending interrupt enable
 - CAN_IT_FMP1: FIFO1 message pending interrupt enable

Return value:

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

[__HAL_CAN_TRANSMIT_STATUS](#)**Description:**

- Check the transmission status of a CAN Frame.

Parameters:

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

Return value:

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

[__HAL_CAN_FIFO_RELEASE](#)**Description:**

- Release the specified receive FIFO.

Parameters:

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

Return value:

- None

[__HAL_CAN_CANCEL_TRANSMIT](#)**Description:**

- Cancel a transmit request.

Parameters:

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

Return value:

- None

_HAL_CAN_DBG_FREEZE

Description:

- Enable or disable the DBG Freeze for CAN.

Parameters:

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

Return value:

- None

CAN Filter FIFO

CAN_FILTER_FIFO0 Filter FIFO 0 assignment for filter x

CAN_FILTER_FIFO1 Filter FIFO 1 assignment for filter x

CAN Filter Mode

CAN_FILTERMODE_IDMASK Identifier mask mode

CAN_FILTERMODE_IDLIST Identifier list mode

CAN Filter Scale

CAN_FILTERSCALE_16BIT Two 16-bit filters

CAN_FILTERSCALE_32BIT One 32-bit filter

CAN Flags

CAN_FLAG_RQCP0 Request MailBox0 flag

CAN_FLAG_RQCP1 Request MailBox1 flag

CAN_FLAG_RQCP2 Request MailBox2 flag

CAN_FLAG_TXOK0 Transmission OK MailBox0 flag

CAN_FLAG_TXOK1 Transmission OK MailBox1 flag

CAN_FLAG_TXOK2 Transmission OK MailBox2 flag

CAN_FLAG_TME0 Transmit mailbox 0 empty flag

CAN_FLAG_TME1 Transmit mailbox 0 empty flag

CAN_FLAG_TME2 Transmit mailbox 0 empty flag

CAN_FLAG_FF0 FIFO 0 Full flag

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

CAN Identifier Type

CAN_ID_STD	Standard Id
CAN_ID_EXT	Extended Id

CAN InitStatus

CAN_INITSTATUS_FAILED	CAN initialization failed
CAN_INITSTATUS_SUCCESS	CAN initialization OK

CAN Interrupts

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

CAN Mailboxes Definition

CAN_TXMAILBOX_0
CAN_TXMAILBOX_1
CAN_TXMAILBOX_2

CAN Operating Mode

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

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

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_STID

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 BRESTOP 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 BRESTOP 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 BRESTOP=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_DelInit\(\)**](#)
- [**HAL_CEC_MspInit\(\)**](#)
- [**HAL_CEC_MspDelInit\(\)**](#)

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

7.2.6 HAL_CEC_DeInit

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

7.2.7 HAL_CEC_MspInit

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

7.2.8 HAL_CEC_MspDeInit

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

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

7.2.9 HAL_CEC_Transmit

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

7.2.10 HAL_CEC_Receive

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

7.2.11 HAL_CEC_Transmit_IT

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

7.2.12 HAL_CEC_Receive_IT

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

7.2.13 HAL_CEC_GetReceivedFrameSize

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

7.2.14 HAL_CEC_IRQHandler

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

7.2.15 HAL_CEC_TxCpltCallback

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

7.2.16 HAL_CEC_RxCpltCallback

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

7.2.17 HAL_CEC_ErrorCallback

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

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

7.2.18 HAL_CEC_GetState

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

7.2.19 HAL_CEC_GetError

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

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-Overrun

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_RXACKE	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_TXACKE	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_TXACKE`: 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_RXACKE`: 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:

- ITStatus

__HAL_CEC_CLEAR_FLAG

Description:

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

Parameters:

- __HANDLE__: specifies the CEC Handle.
- __FLAG__: specifies the interrupt/status flag to clear. This parameter can be one of the following values:
 - CEC_FLAG_TXACKE: 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_RXBR: Rx-Byte Request.
 - CEC_FLAG_ARBLST: Arbitration Lost
 - CEC_FLAG_RXACKE: 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

__HAL_CEC_ENABLE_IT

Description:

- Enables the specified CEC interrupt.

Parameters:

- [HANDLE](#): specifies the CEC Handle.
- [INTERRUPT](#): specifies the CEC interrupt to enable. This parameter can be one of the following values:
 - CEC_IT_TXACKE: 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_RXACKE: 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_DISABLE_IT](#)**Description:**

- Disables the specified CEC interrupt.

Parameters:

- [HANDLE](#): specifies the CEC Handle.
- [INTERRUPT](#): specifies the CEC interrupt to disable. This parameter can be one of the following values:
 - CEC_IT_TXACKE: 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_RXACKE: 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_RXACKE: Rx-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_RXACKE: 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

_HAL_CEC_ENABLE**Description:**

- Enables the CEC device.

Parameters:

- _HANDLE: specifies the CEC Handle.

Return value:

- none

_HAL_CEC_DISABLE**Description:**

- Disables the CEC device.

Parameters:

- _HANDLE: specifies the CEC Handle.

Return value:

- none

_HAL_CEC_FIRST_BYTE_TX_SET**Description:**

- Set Transmission Start flag.

Parameters:

- _HANDLE: specifies the CEC Handle.

Return value:

- none

_HAL_CEC_LAST_BYTE_TX_SET**Description:**

- Set Transmission End flag.

Parameters:

- _HANDLE: specifies the CEC Handle.

Return value:

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

`__HAL_CEC_GET_TRANSMISSION_START_FLAG`

G

Description:

- Get Transmission Start flag.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.

Return value:

- FlagStatus

`__HAL_CEC_GET_TRANSMISSION_END_FLAG`

G

Description:

- Get Transmission End flag.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.

Return value:

- FlagStatus

`__HAL_CEC_CLEAR_OAR`

G

Description:

- Clear OAR register.

Parameters:

- `__HANDLE__`: specifies the CEC Handle.

Return value:

- none

`__HAL_CEC_SET_OAR`

G

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_RXBR

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_RXBR

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_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_MPUMemoryAccessCacheable**
- **`uint8_t MPU_Region_InitTypeDef::IsBufferable`**
Specifies the bufferable status of the protected region. This parameter can be a value of **CORTEX_MPUMemoryAccessBufferable**

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 programing manual for details in how to configure priority. When the `NVIC_PRIORITYGROUP_0` is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the sub priority. IRQ priority order (sorted by highest to lowest priority): Lowest preemption priority Lowest sub priority Lowest hardware priority (IRQ number)

How to configure Systick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The `HAL_SYSTICK_Config()` function calls the `SysTick_Config()` function which is a CMSIS function that:
 - Configures the SysTick Reload register with value passed as function parameter.
 - Configures the SysTick IRQ priority to the lowest value (0x0F).
 - Resets the SysTick Counter register.
 - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
 - Enables the SysTick Interrupt.
 - Starts the SysTick Counter.
- You can change the SysTick Clock source to be `HCLK_Div8` by calling the macro `__HAL_CORTEX_SYSTICKCLK_CONFIG(SYSTICK_CLKSOURCE_HCLK_DIV8)` just after the `HAL_SYSTICK_Config()` function call. The `__HAL_CORTEX_SYSTICKCLK_CONFIG()` macro is defined inside the `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 0xFFFFFFF

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	void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)
Function Description	Sets the priority grouping field (preemption priority and subpriority) using the required unlock sequence.
Parameters	<ul style="list-style-type: none"> • PriorityGroup: 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
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • When the NVIC_PriorityGroup_0 is selected, IRQ preemption is no more possible. The pending IRQ priority will be managed only by the subpriority.

8.2.5 HAL_NVIC_SetPriority

Function Name	void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)
Function Description	Sets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f7xxxx.h)) PreemptPriority: 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 SubPriority: the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.
Return values	<ul style="list-style-type: none"> None

8.2.6 HAL_NVIC_EnableIRQ

Function Name	void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)
Function Description	Enables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f7xxxx.h))
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

8.2.7 HAL_NVIC_DisableIRQ

Function Name	void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)
Function Description	Disables a device specific interrupt in the NVIC interrupt controller.
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f7xxxx.h))
Return values	<ul style="list-style-type: none"> None

8.2.8 HAL_NVIC_SystemReset

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

8.2.9 HAL_SYSTICK_Config

Function Name	uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)
Function Description	Initializes the System Timer and its interrupt, and starts the System

Tick Timer.

Parameters	<ul style="list-style-type: none"> TicksNumb: Specifies the ticks Number of ticks between two interrupts.
Return values	<ul style="list-style-type: none"> status - 0 Function succeeded. 1 Function failed.

8.2.10 HAL_MPU_ConfigRegion

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

8.2.11 HAL_NVIC_GetPriorityGrouping

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

8.2.12 HAL_NVIC_GetPriority

Function Name	void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)
Function Description	Gets the priority of an interrupt.
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f7xxxx.h)) PriorityGroup: 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 pPreemptPriority: Pointer on the Preemptive priority value (starting from 0). pSubPriority: Pointer on the Subpriority value (starting from 0).
Return values	<ul style="list-style-type: none"> None

8.2.13 HAL_NVIC_SetPendingIRQ

Function Name	void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)
Function Description	Sets Pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f7xxxx.h))
Return values	<ul style="list-style-type: none"> None
8.2.14 HAL_NVIC_GetPendingIRQ	
Function Name	uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)
Function Description	Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f7xxxx.h))
Return values	<ul style="list-style-type: none"> status - 0 Interrupt status is not pending. 1 Interrupt status is pending.
8.2.15 HAL_NVIC_ClearPendingIRQ	
Function Name	void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)
Function Description	Clears the pending bit of an external interrupt.
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f7xxxx.h))
Return values	<ul style="list-style-type: none"> None
8.2.16 HAL_NVIC_GetActive	
Function Name	uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)
Function Description	Gets active interrupt (reads the active register in NVIC and returns the active bit).
Parameters	<ul style="list-style-type: none"> IRQn: External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32f7xxxx.h))
Return values	<ul style="list-style-type: none"> status - 0 Interrupt status is not pending. 1 Interrupt status is pending.
8.2.17 HAL_SYSTICK_CLKSourceConfig	
Function Name	void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)
Function Description	Configures the SysTick clock source.
Parameters	<ul style="list-style-type: none"> CLKSource: specifies the SysTick clock source. This

parameter can be one of the following values:
SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.
SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.

Return values • None

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_CON FIG **Description:**
 • Configures the SysTick clock source.
Parameters:
 • **_CLKSRC_**: specifies the SysTick clock source. This parameter can be one of the following values:
 – **SYSTICK_CLKSOURCE_HCLK_DIV8**: AHB clock divided by 8 selected as SysTick clock source.
 – **SYSTICK_CLKSOURCE_HCLK**: AHB clock selected as SysTick clock source.

Return value:

- None

CORTEX MPU Instruction Access Bufferable

MPU_ACCESS_BUFFERABLE

MPU_ACCESS_NOT_BUFFERABLE

CORTEX MPU Instruction Access Cacheable

MPU_ACCESS_CACHEABLE

MPU_ACCESS_NOT_CACHEABLE

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_SHAREABLE
MPU_ACCESS_NOT_SHAREABLE
MPU HFNMI and PRIVILEGED Access control
MPU_HFNMI_PRIVDEF_NONE
MPU_HARDFAULT_NMI
MPU_PRIVILEGED_DEFAULT
MPU_HFNMI_PRIVDEF
CORTEX MPU Instruction Access
MPU_INSTRUCTION_ACCESS_ENABLE
MPU_INSTRUCTION_ACCESS_DISABLE
CORTEX MPU Region Enable
MPU_REGION_ENABLE
MPU_REGION_DISABLE
CORTEX MPU Region Number
MPU_REGION_NUMBER0
MPU_REGION_NUMBER1
MPU_REGION_NUMBER2
MPU_REGION_NUMBER3
MPU_REGION_NUMBER4
MPU_REGION_NUMBER5
MPU_REGION_NUMBER6
MPU_REGION_NUMBER7
CORTEX MPU Region Permission Attributes
MPU_REGION_NO_ACCESS
MPU_REGION_PRIV_RW
MPU_REGION_PRIV_RW_URO
MPU_REGION_FULL_ACCESS
MPU_REGION_PRIV_RO
MPU_REGION_PRIV_RO_URO
CORTEX MPU Region Size
MPU_REGION_SIZE_32B
MPU_REGION_SIZE_64B
MPU_REGION_SIZE_128B
MPU_REGION_SIZE_256B
MPU_REGION_SIZE_512B
MPU_REGION_SIZE_1KB

MPU_REGION_SIZE_2KB
MPU_REGION_SIZE_4KB
MPU_REGION_SIZE_8KB
MPU_REGION_SIZE_16KB
MPU_REGION_SIZE_32KB
MPU_REGION_SIZE_64KB
MPU_REGION_SIZE_128KB
MPU_REGION_SIZE_256KB
MPU_REGION_SIZE_512KB
MPU_REGION_SIZE_1MB
MPU_REGION_SIZE_2MB
MPU_REGION_SIZE_4MB
MPU_REGION_SIZE_8MB
MPU_REGION_SIZE_16MB
MPU_REGION_SIZE_32MB
MPU_REGION_SIZE_64MB
MPU_REGION_SIZE_128MB
MPU_REGION_SIZE_256MB
MPU_REGION_SIZE_512MB
MPU_REGION_SIZE_1GB
MPU_REGION_SIZE_2GB
MPU_REGION_SIZE_4GB

MPU TEX Levels

MPU_TEX_LEVEL0
MPU_TEX_LEVEL1
MPU_TEX_LEVEL2

CORTEX Preemption Priority Group

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

CORTEX Private Macros

IS_NVIC_PRIORITY_GROUP
IS_NVIC_PREEMPTION_PRIORITY
IS_NVIC_SUB_PRIORITY

IS_NVIC_DEVICE_IRQ
IS_SYSTICK_CLK_SOURCE
IS MPU_REGION_ENABLE
IS MPU_INSTRUCTION_ACCESS
IS MPU_ACCESS_SHAREABLE
IS MPU_ACCESS_CACHEABLE
IS MPU_ACCESS_BUFFERABLE
IS MPU_TEX_LEVEL
IS MPU_REGION_PERMISSION_ATTRIBUTE
IS MPU_REGION_NUMBER
IS MPU_REGION_SIZE
IS MPU_SUB_REGION_DISABLE
CORTEX_SysTick clock source
SYSTICK_CLKSOURCE_HCLK_DIV8
SYSTICK_CLKSOURCE_HCLK

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_DelInit\(\)*](#)
- [*HAL_CRC_MspInit\(\)*](#)
- [*HAL_CRC_MspDelInit\(\)*](#)

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	<code>HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)</code>
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">• hcrc: CRC handle
Return values	<ul style="list-style-type: none">• HAL status

9.2.6 HAL_CRC_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_CRC_DelInit (CRC_HandleTypeDef * hcrc)</code>
---------------	---

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

9.2.7 HAL_CRC_Msplnit

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

9.2.8 HAL_CRC_MspDelnit

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

9.2.9 HAL_CRC_Accumulate

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

9.2.10 HAL_CRC_Calculate

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

9.2.11 HAL_CRC_GetState

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

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.

Parameters:

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

Return value:

- None.

`__HAL_CRC_SET_IDR`**Description:**

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

Parameters:

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

Return value:

- None

`__HAL_CRC_GET_IDR`**Description:**

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

Parameters:

- `__HANDLE__`: CRC handle

Return value:

- 8-bit: value of the ID register

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

10.1.3 [*HAL_CRCEx_Input_Data_Reverse*](#)

Function Name	<code>HAL_StatusTypeDef HAL_CRCEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)</code>
Function Description	Set the Reverse Input data mode.
Parameters	<ul style="list-style-type: none"> • hcrc: CRC handle • InputReverseMode: 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

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 CRCEEx Firmware driver defines

10.2.1 CRCEEx

CRC Extended exported macros

<code>_HAL_CRC_OUTPUTREVERSAL_ENABLE</code>	Description: <ul style="list-style-type: none"> Set CRC output reversal. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: CRC handle Return value: <ul style="list-style-type: none"> None.
<code>_HAL_CRC_OUTPUTREVERSAL_DISABLE</code>	Description: <ul style="list-style-type: none"> Unset CRC output reversal. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: CRC handle Return value: <ul style="list-style-type: none"> None.
<code>_HAL_CRC_POLYNOMIAL_CONFIG</code>	Description: <ul style="list-style-type: none"> Set CRC non-default polynomial. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: CRC handle <code>_POLYNOMIAL_</code>: 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 CRYP Generic Driver

11.1 CRYP Firmware driver registers structures

11.1.1 CRYP_InitTypeDef

Data Fields

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

Field Documentation

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

11.1.2 CRYP_HandleTypeDefDef

Data Fields

- ***CRYP_TypeDef * Instance***
- ***CRYP_InitTypeDef Init***
- ***uint8_t * pCrypInBuffPtr***
- ***uint8_t * pCrypOutBuffPtr***
- ***_IO uint16_t CrypInCount***
- ***_IO uint16_t CrypOutCount***
- ***HAL_StatusTypeDef Status***
- ***HAL_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***_IO HAL_CRYP_STATETypeDef State***

Field Documentation

- ***CRYP_TypeDef* CRYP_HandleTypeDef::Instance***
CRYP registers base address
- ***CRYP_InitTypeDef CRYP_HandleTypeDef::Init***
CRYP required parameters
- ***uint8_t* CRYP_HandleTypeDef::pCrypInBuffPtr***
Pointer to CRYP processing (encryption, decryption,...) buffer
- ***uint8_t* CRYP_HandleTypeDef::pCrypOutBuffPtr***
Pointer to CRYP processing (encryption, decryption,...) buffer
- ***_IO uint16_t CRYP_HandleTypeDef::CrypInCount***
Counter of inputed data
- ***_IO uint16_t CRYP_HandleTypeDef::CrypOutCount***
Counter of output data
- ***HAL_StatusTypeDef CRYP_HandleTypeDef::Status***
CRYP peripheral status
- ***HAL_PhaseTypeDef CRYP_HandleTypeDef::Phase***
CRYP peripheral phase
- ***DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmain***
CRYP In DMA handle parameters
- ***DMA_HandleTypeDef* CRYP_HandleTypeDef::hdmaout***
CRYP Out DMA handle parameters
- ***HAL_LockTypeDef CRYP_HandleTypeDef::Lock***
CRYP locking object
- ***_IO HAL_CRYP_STATETypeDef CRYP_HandleTypeDef::State***
CRYP peripheral state

11.2 CRYP Firmware driver API description

11.2.1 How to use this driver

The CRYP HAL driver can be used as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
 - a. Enable the CRYP interface clock using `_HAL_RCC_CRYP_CLK_ENABLE()`
 - b. In case of using interrupts (e.g. `HAL_CRYP_AESECB_Encrypt_IT()`)
 - Configure the CRYP interrupt priority using `HAL_NVIC_SetPriority()`
 - Enable the CRYP IRQ handler using `HAL_NVIC_EnableIRQ()`
 - In CRYP IRQ handler, call `HAL_CRYP_IRQHandler()`

- c. In case of using DMA to control data transfer (e.g. HAL_CRYP_AESECB_Encrypt_DMA())
 - Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
 - Configure and enable two DMA streams one for managing data transfer from memory to peripheral (input stream) and another stream for managing data transfer from peripheral to memory (output stream)
 - Associate the initialized DMA handle to the CRYP DMA handle using __HAL_LINKDMA()
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
- 2. Initialize the CRYP HAL using HAL_CRYP_Init(). This function configures mainly:
 - a. The data type: 1-bit, 8-bit, 16-bit and 32-bit
 - b. The key size: 128, 192 and 256. This parameter is relevant only for AES
 - c. The encryption/decryption key. It's size depends on the algorithm used for encryption/decryption
 - d. The initialization vector (counter). It is not used ECB mode.
- 3. Three processing (encryption/decryption) functions are available:
 - a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished, e.g. HAL_CRYP_AESCBC_Encrypt()
 - b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_CRYP_AESCBC_Encrypt_IT()
 - c. DMA mode: encryption and decryption APIs are not blocking functions i.e. the data transfer is ensured by DMA, e.g. HAL_CRYP_AESCBC_Encrypt_DMA()
- 4. When the processing function is called at first time after HAL_CRYP_Init() the CRYP peripheral is initialized and processes the buffer in input. At second call, the processing function performs an append of the already processed buffer. When a new data block is to be processed, call HAL_CRYP_Init() then the processing function.
- 5. Call HAL_CRYP_DelInit() to deinitialize the CRYP peripheral.

11.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRYP according to the specified parameters in the CRYP_InitTypeDef and creates the associated handle
- Deinitialize the CRYP peripheral
- Initialize the CRYP MSP
- Deinitialize CRYP MSP

This section contains the following APIs:

- [**HAL_CRYP_Init\(\)**](#)
- [**HAL_CRYP_DelInit\(\)**](#)
- [**HAL_CRYP_MspInit\(\)**](#)
- [**HAL_CRYP_MspDelInit\(\)**](#)

11.2.3 AES processing functions

This section provides functions allowing to:

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

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode

This section contains the following APIs:

- [*HAL_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 ciphertext using TDES based on ECB or CBC chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode

This section contains the following APIs:

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

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_CRYP_InCpltCallback()`
- `HAL_CRYP_OutCpltCallback()`
- `HAL_CRYP_ErrorCallback()`

11.2.7 CRYP IRQ handler management

This section provides CRYP IRQ handler function.

This section contains the following APIs:

- `HAL_CRYP_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_CRYP_GetState()`

11.2.9 HAL_CRYP_Init

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

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

Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> HAL status

11.2.10 HAL_CRYP_DelInit

Function Name	HAL_StatusTypeDef HAL_CRYP_DelInit (CRYP_HandleTypeDef * hcryp)
Function Description	Deinitializes the CRYP peripheral.
Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none"> HAL status

11.2.11 HAL_CRYP_MspInit

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

11.2.12 HAL_CRYP_MspDelInit

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

11.2.13 HAL_CRYP_AESECB_Encrypt

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

11.2.14 HAL_CRYP_AESCBC_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt
---------------	--

(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)

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

11.2.15 HAL_CRYP_AESCTR_Encrypt

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

11.2.16 HAL_CRYP_AESECB_Decrypt

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

11.2.17 HAL_CRYP_AESCBC_Decrypt

Function Name	HAL_StatusTypeDef HAL_CRYP_AESCBC_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYP peripheral in AES CBC decryption mode then

decrypted pCypherData.

Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module pCypherData: Pointer to the ciphertext buffer Size: Length of the plaintext buffer, must be a multiple of 16. pPlainData: Pointer to the plaintext buffer Timeout: Specify Timeout value
Return values	<ul style="list-style-type: none"> HAL status

11.2.18 HAL_CRYP_AESCTR_Decrypt

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

11.2.19 HAL_CRYP_AESECB_Encrypt_IT

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

11.2.20 HAL_CRYP_AESCBC_Encrypt_IT

Function Name	HAL_StatusTypeDef HAL_CRYP_AESCBC_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYP peripheral in AES CBC encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module pPlainData: Pointer to the plaintext buffer

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

Return values

11.2.21 HAL_CRYP_AESCTR_Encrypt_IT

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

11.2.22 HAL_CRYP_AESECB_Decrypt_IT

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

11.2.23 HAL_CRYP_AESCBC_Decrypt_IT

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

11.2.24 HAL_CRYP_AESCTR_Decrypt_IT

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

11.2.25 HAL_CRYP_AESECB_Encrypt_DMA

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

11.2.26 HAL_CRYP_AESCBC_Encrypt_DMA

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

11.2.27 HAL_CRYP_AESCTR_Encrypt_DMA

Function Name	<code>HAL_StatusTypeDef HAL_CRYP_AESCTR_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)</code>
Function Description	Initializes the CRYP peripheral in AES CTR encryption mode using

DMA.

- | | |
|---------------|---|
| Parameters | <ul style="list-style-type: none"> • hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pCypherData: Pointer to the ciphertext buffer • Size: Length of the plaintext buffer, must be a multiple of 16. • pPlainData: Pointer to the plaintext buffer |
| Return values | <ul style="list-style-type: none"> • HAL status |

11.2.28 HAL_CRYP_AESECB_Decrypt_DMA

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

11.2.29 HAL_CRYP_AESCBC_Decrypt_DMA

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

11.2.30 HAL_CRYP_AESCTR_Decrypt_DMA

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

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

Return values • HAL status

11.2.31 HAL_CRYP_DESECB_Encrypt

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

11.2.32 HAL_CRYP_DESECB_Decrypt

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

11.2.33 HAL_CRYP_DESCBC_Encrypt

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

11.2.34 HAL_CRYP_DESCBC_Decrypt

Function Name	HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt (CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t
---------------	--

Size, uint8_t * pCypherData, uint32_t Timeout)

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

11.2.35 HAL_CRYP_DESECB_Encrypt_IT

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

11.2.36 HAL_CRYP_DESCBC_Encrypt_IT

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

11.2.37 HAL_CRYP_DESECB_Decrypt_IT

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

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

Return values

11.2.38 HAL_CRYP_DESCBC_Decrypt_IT

Function Name

**HAL_StatusTypeDef HAL_CRYP_DESCBC_Decrypt_IT
(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,
uint16_t Size, uint8_t * pPlainData)**

Function Description

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

Parameters

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

Return values

- HAL status

11.2.39 HAL_CRYP_DESECB_Encrypt_DMA

Function Name

**HAL_StatusTypeDef HAL_CRYP_DESECB_Encrypt_DMA
(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t
Size, uint8_t * pCypherData)**

Function Description

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

Parameters

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

Return values

- HAL status

11.2.40 HAL_CRYP_DESCBC_Encrypt_DMA

Function Name

**HAL_StatusTypeDef HAL_CRYP_DESCBC_Encrypt_DMA
(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t
Size, uint8_t * pCypherData)**

Function Description

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

Parameters

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

Return values

- HAL status

11.2.41 HAL_CRYP_DESECB_Decrypt_DMA

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

11.2.42 HAL_CRYP_DESCBC_Decrypt_DMA

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

11.2.43 HAL_CRYP_TDESECB_Encrypt

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

11.2.44 HAL_CRYP_TDESECB_Decrypt

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

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

11.2.45 HAL_CRYP_TDESCBC_Encrypt

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

11.2.46 HAL_CRYP_TDESCBC_Decrypt

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

11.2.47 HAL_CRYP_TDESECB_Encrypt_IT

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

- **pCypherData:** Pointer to the ciphertext buffer
- HAL status

11.2.48 HAL_CRYP_TDESCBC_Encrypt_IT

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

11.2.49 HAL_CRYP_TDESECB_Decrypt_IT

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

11.2.50 HAL_CRYP_TDESCBC_Decrypt_IT

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

11.2.51 HAL_CRYP_TDESECB_Encrypt_DMA

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

using DMA.

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

11.2.52 HAL_CRYP_TDESCBC_Encrypt_DMA

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

11.2.53 HAL_CRYP_TDESECB_Decrypt_DMA

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

11.2.54 HAL_CRYP_TDESCBC_Decrypt_DMA

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

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

11.2.55 HAL_CRYP_InCpltCallback

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

11.2.56 HAL_CRYP_OutCpltCallback

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

11.2.57 HAL_CRYP_ErrorCallback

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

11.2.58 HAL_CRYP_IRQHandler

Function Name	void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)
Function Description	This function handles CRYP interrupt request.
Parameters	<ul style="list-style-type: none">• hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none">• None

11.2.59 HAL_CRYP_GetState

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

`__HAL_CRYP_RESET_HANDLE_STATE`

Description:

- Reset CRYP handle state.

Parameters:

- `__HANDLE__`: specifies the CRYP handle.

Return value:

- None

`__HAL_CRYP_ENABLE`

Description:

- Enable/Disable the CRYP peripheral.

Parameters:

- `__HANDLE__`: specifies the CRYP handle.

Return value:

- None

`__HAL_CRYP_DISABLE`

`__HAL_CRYP_FIFO_FLUSH`

Description:

- Flush the data FIFO.

Parameters:

- `__HANDLE__`: specifies the CRYP handle.

Return value:

- None

`__HAL_CRYP_SET_MODE`

Description:

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

Parameters:

- `__HANDLE__`: specifies the CRYP handle.
- MODE: The algorithm mode.

Return value:

- None

`__HAL_CRYP_GET_FLAG`

Description:

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

Parameters:

- `__HANDLE__`: specifies the CRYP handle.
- `__FLAG__`: specifies the flag to check.
This parameter can be one of the following values:

- 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_OFU: Output FIFO is full
- CRYPT_FLAG_OUTRIS: Input FIFO service raw interrupt is pending

Return value:

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

[__HAL_CRYPT_GET_IT](#)**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).

[__HAL_CRYPT_ENABLE_IT](#)**Description:**

- Enable the CRYPT interrupt.

Parameters:

- __HANDLE__: specifies the CRYPT handle.
- __INTERRUPT__: CRYPT Interrupt.

Return value:

- None

[__HAL_CRYPT_DISABLE_IT](#)**Description:**

- Disable the CRYPT interrupt.

Parameters:

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

Return value:

- None

CRYP Key Size

`CRYP_KEYSIZE_128B`

`CRYP_KEYSIZE_192B`

`CRYP_KEYSIZE_256B`

CRYP Private Constants

`CRYP_FLAG_MASK`

CRYP_Private_define

`CRYP_TIMEOUT_VALUE`

CRYP Private Macros

`IS_CRYP_KEYSIZE`

`IS_CRYP_DATATYPE`

12 HAL CRYP Extension Driver

12.1 CRYPEx Firmware driver API description

12.1.1 How to use this driver

The CRYP Extension HAL driver can be used as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
 - a. Enable the CRYP interface clock using __HAL_RCC_CRYP_CLK_ENABLE()
 - b. In case of using interrupts (e.g. HAL_CRYPEx_AESGCM_Encrypt_IT())
 - Configure the CRYP interrupt priority using HAL_NVIC_SetPriority()
 - Enable the CRYP IRQ handler using HAL_NVIC_EnableIRQ()
 - In CRYP IRQ handler, call HAL_CRYP_IRQHandler()
 - c. In case of using DMA to control data transfer (e.g. HAL_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 CRYP DMA handle using __HAL_LINKDMA()
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA Streams. The output stream should have higher priority than the input stream HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the CRYP HAL using HAL_CRYP_Init(). This function configures mainly:
 - a. The data type: 1-bit, 8-bit, 16-bit and 32-bit
 - b. The key size: 128, 192 and 256. This parameter is relevant only for AES
 - c. The encryption/decryption key. Its size depends on the algorithm used for encryption/decryption
 - d. The initialization vector (counter). It is not used ECB mode.
3. Three processing (encryption/decryption) functions are available:
 - a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished e.g. HAL_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_CRYP_Init() the CRYP peripheral is initialized and processes the buffer in input. At second call, the processing function performs an append of the already processed buffer. When a new data block is to be processed, call HAL_CRYP_Init() then the processing function.
5. 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_CRYP_DelInit() to deinitialize the CRYP 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 ciphertext 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	<code>HAL_StatusTypeDef HAL_CRYPEX_AESCCM_Encrypt(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)</code>
Function Description	Initializes the CRYP peripheral in AES CCM encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module • pPlainData: Pointer to the plaintext buffer • Size: Length of the plaintext buffer, must be a multiple of 16 • pCypherData: Pointer to the ciphertext buffer • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

12.1.5 HAL_CRYPEx_AESGCM_Encrypt

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

12.1.6 HAL_CRYPEx_AESGCM_Decrypt

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

12.1.7 HAL_CRYPEx_AESGCM_Finish

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

12.1.8 HAL_CRYPEx_AESCCM_Finish

Function Name	<code>HAL_StatusTypeDef HAL_CRYPEx_AESCCM_Finish (CRYP_HandleTypeDef * hcryp, uint8_t * AuthTag, uint32_t Timeout)</code>
---------------	---

Function Description	Computes the authentication TAG for AES CCM mode.
Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module AuthTag: Pointer to the authentication buffer Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> This API is called after HAL_AES_CCM_Encrypt() / HAL_AES_CCM_Decrypt()

12.1.9 HAL_CRYPEx_AESCCM_Decrypt

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

12.1.10 HAL_CRYPEx_AESGCM_Encrypt_IT

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

12.1.11 HAL_CRYPEx_AESCCM_Encrypt_IT

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

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

Return values

12.1.12 HAL_CRYPEx_AESGCM_Decrypt_IT

Function Name **HAL_StatusTypeDef HAL_CRYPEx_AESGCM_Decrypt_IT
(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,
uint16_t Size, uint8_t * pPlainData)**

Function Description Initializes the CRYP peripheral in AES GCM decryption mode using IT.

- Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
 - **pCypherData:** Pointer to the ciphertext buffer
 - **Size:** Length of the ciphertext buffer, must be a multiple of 16
 - **pPlainData:** Pointer to the plaintext buffer

Return values

- HAL status

12.1.13 HAL_CRYPEx_AESCCM_Decrypt_IT

Function Name **HAL_StatusTypeDef HAL_CRYPEx_AESCCM_Decrypt_IT
(CRYP_HandleTypeDef * hcryp, uint8_t * pCypherData,
uint16_t Size, uint8_t * pPlainData)**

Function Description Initializes the CRYP peripheral in AES CCM decryption mode using interrupt then decrypted pCypherData.

- Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
 - **pCypherData:** Pointer to the ciphertext buffer
 - **Size:** Length of the plaintext buffer, must be a multiple of 16
 - **pPlainData:** Pointer to the plaintext buffer

Return values

- HAL status

12.1.14 HAL_CRYPEx_AESGCM_Encrypt_DMA

Function Name **HAL_StatusTypeDef HAL_CRYPEx_AESGCM_Encrypt_DMA
(CRYP_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t
Size, uint8_t * pCypherData)**

Function Description Initializes the CRYP peripheral in AES GCM encryption mode using DMA.

- Parameters
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
 - **pPlainData:** Pointer to the plaintext buffer
 - **Size:** Length of the plaintext buffer, must be a multiple of 16
 - **pCypherData:** Pointer to the ciphertext buffer

Return values

- HAL status

12.1.15 HAL_CRYPEx_AESCCM_Encrypt_DMA

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

12.1.16 HAL_CRYPEEx_AESGCM_Decrypt_DMA

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

12.1.17 HAL_CRYPEEx_AESCCM_Decrypt_DMA

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

12.1.18 HAL_CRYPEEx_GCMCCM_IRQHandler

Function Name	<code>void HAL_CRYPEEx_GCMCCM_IRQHandler(CRYP_HandleTypeDef * hcryp)</code>
Function Description	This function handles CRYPEEx interrupt request.
Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYPEEx_HandleTypeDef structure that contains the configuration information for CRYP module

- 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, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

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

DMA1 requests are mapped as following:

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

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_DelInit\(\)*](#)
- [*HAL_DAC_MspInit\(\)*](#)
- [*HAL_DAC_MspDelInit\(\)*](#)

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

13.2.8 HAL_DAC_DeInit

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

13.2.9 HAL_DAC_MspInit

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

13.2.10 HAL_DAC_MspDeInit

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

13.2.11 HAL_DAC_Start

Function Name	HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef *hdac, uint32_t Channel)
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> 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
Return values	<ul style="list-style-type: none"> HAL status

13.2.12 HAL_DAC_Stop

Function Name	HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef *hdac, uint32_t Channel)
Function Description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none"> 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
Return values	<ul style="list-style-type: none"> HAL status

13.2.13 HAL_DAC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> 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 pData: The destination peripheral Buffer address. Length: The length of data to be transferred from memory to DAC peripheral Alignment: Specifies the data alignment for DAC channel. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selectedDAC_ALIGN_12B_L: 12bit left data alignment selectedDAC_ALIGN_12B_R: 12bit right data alignment selected
Return values	<ul style="list-style-type: none"> HAL status

13.2.14 HAL_DAC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function Description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none"> 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
Return values	<ul style="list-style-type: none"> HAL status

13.2.15 HAL_DAC_GetValue

Function Name	uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"> 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
Return values	<ul style="list-style-type: none"> The selected DAC channel data output value.

13.2.16 HAL_DAC_IRQHandler

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

13.2.17 HAL_DAC_ConvCpltCallbackCh1

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

13.2.18 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"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

13.2.19 HAL_DAC_ErrorCallbackCh1

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

13.2.20 HAL_DAC_DMAUnderrunCallbackCh1

Function Name	<code>void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)</code>
Function Description	DMA underrun DAC callback for channel1.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

13.2.21 HAL_DAC_ConfigChannel

Function Name	<code>HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)</code>
Function Description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. sConfig: DAC configuration structure. Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> HAL status

13.2.22 HAL_DAC_SetValue

Function Name	<code>HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)</code>
Function Description	Set the specified data holding register value for DAC channel.
Parameters	<ul style="list-style-type: none"> 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
 - **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 | <ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. |
| Return values | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. |
| Return values | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. |
| Return values | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. |
| Return values | <ul style="list-style-type: none"> • 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"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None

13.2.28 HAL_DAC_ErrorCallbackCh1

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

13.2.29 HAL_DAC_DMAUnderrunCallbackCh1

Function Name	void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)
Function Description	DMA underrun DAC callback for channel1.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None

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_DMAUNDERUNCH1	DAC channel1 DAM underrun error
-----------------------------	---------------------------------

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

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

DAC Exported Macros

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

- Reset DAC handle state.

Parameters:

- `__HANDLE__`: specifies the DAC handle.

Return value:

- None

`__HAL_DAC_ENABLE`

Description:

- Enable the DAC channel.

Parameters:

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

Return value:

- None

`__HAL_DAC_DISABLE`

Description:

- Disable the DAC channel.

Parameters:

- `__HANDLE__`: specifies the DAC handle
- `__DAC_CHANNEL__`: specifies the DAC channel

Return value:

- None

`__HAL_DAC_ENABLE_IT`

Description:

- Enable the DAC interrupt.

Parameters:

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

Return value:

- None

`__HAL_DAC_DISABLE_IT`

Description:

- Disable the DAC interrupt.

Parameters:

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

Return value:

- None

`__HAL_DAC_GET_IT_SOURCE`

Description:

- Checks if the specified DAC interrupt

source is enabled or disabled.

Parameters:

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

Return value:

- State: of interruption (SET or RESET)

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

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

14.1.4 HAL_DACEx_TriangleWaveGenerate

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

Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

14.1.5 HAL_DACEx_NoiseWaveGenerate

Function Name	<code>HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate(DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)</code>
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> 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 | <ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • 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"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None
14.1.9 HAL_DACEx_ErrorCallbackCh2	
Function Name	void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)
Function Description	Error DAC callback for Channel2.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None
14.1.10 HAL_DACEx_DMAUnderrunCallbackCh2	
Function Name	void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)
Function Description	DMA underrun DAC callback for channel2.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None

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
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	HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef *hdcmi)
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"> • hdcmi: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	<ul style="list-style-type: none"> • HAL status

15.2.7 HAL_DCMI_DelInit

Function Name	HAL_StatusTypeDef HAL_DCMI_DelInit (DCMI_HandleTypeDef *hdcmi)
Function Description	Deinitializes the DCMI peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> • hdcmi: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	<ul style="list-style-type: none"> • HAL status

15.2.8 HAL_DCMI_MspInit



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

15.2.9 HAL_DCMI_MspDeInit

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

15.2.10 HAL_DCMI_Start_DMA

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

15.2.11 HAL_DCMI_Stop

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

15.2.12 HAL_DCMI_IRQHandler

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

15.2.13 HAL_DCMI_ErrorCallback

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

15.2.14 HAL_DCMI_LineEventCallback

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

15.2.15 HAL_DCMI_VsyncEventCallback

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

15.2.16 HAL_DCMI_FrameEventCallback

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

15.2.17 HAL_DCMI_ConfigCROP

Function Name	<code>HAL_StatusTypeDef HAL_DCMI_ConfigCROP (DCMI_HandleTypeDef * hdcmi, uint32_t X0, uint32_t Y0, uint32_t XSize, uint32_t YSize)</code>
Function Description	Configure the DCMI CROP coordinate.
Parameters	<ul style="list-style-type: none"> • hdcmi: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI. • YSize: DCMI Line number • XSize: DCMI Pixel per line • X0: DCMI window X offset

- **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	<ul style="list-style-type: none"> • hdcmi: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hdcmi: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hdcmi: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hdcmi: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	<ul style="list-style-type: none"> • 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

<code>_HAL_DCMI_RESET_HANDLE_STATE</code>	Description: • Reset DCMI handle state. Parameters: • <code>_HANDLE_</code> : specifies the DCMI handle. Return value: • None
<code>_HAL_DCMI_ENABLE</code>	Description: • Enable the DCMI. Parameters: • <code>_HANDLE_</code> : DCMI handle Return value: • None
<code>_HAL_DCMI_DISABLE</code>	Description: • Disable the DCMI. Parameters: • <code>_HANDLE_</code> : DCMI handle Return value: • None
<code>_HAL_DCMI_GET_FLAG</code>	Description: • Get the DCMI pending flags. Parameters: • <code>_HANDLE_</code> : DCMI handle • <code>_FLAG_</code> : Get the specified flag. This parameter can be any combination of the following values: – <code>DCMI_FLAG_FRAMERI</code> : Frame

- capture complete flag mask
- DCMI_FLAG_OVFRI: Overflow flag mask
- DCMI_FLAG_ERRRI: Synchronization error flag mask
- DCMI_FLAG_VSYNCRI: VSYNC flag mask
- DCMI_FLAG_LINERI: Line flag mask

Return value:

- The state of FLAG.

__HAL_DCMI_CLEAR_FLAG

Description:

- Clear the DCMI pending flags.

Parameters:

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

Return value:

- None

__HAL_DCMI_ENABLE_IT

Description:

- Enable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
 - DCMI_IT_FRAME: Frame capture complete interrupt mask
 - DCMI_IT_OVF: Overflow interrupt mask
 - DCMI_IT_ERR: Synchronization error interrupt mask
 - DCMI_IT_VSYNC: VSYNC interrupt mask
 - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

_HAL_DCMI_DISABLE_IT**Description:**

- Disable the specified DCMI interrupts.

Parameters:

- HANDLE: DCMI handle
- INTERRUPT: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
 - DCMI_IT_FRAME: Frame capture complete interrupt mask
 - DCMI_IT_OVF: Overflow interrupt mask
 - DCMI_IT_ERR: Synchronization error interrupt mask
 - DCMI_IT_VSYNC: VSYNC interrupt mask
 - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

_HAL_DCMI_GET_IT_SOURCE**Description:**

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

Parameters:

- HANDLE: DCMI handle
- INTERRUPT: specifies the DCMI interrupt source to check. This parameter can be one of the following values:
 - DCMI_IT_FRAME: Frame capture complete interrupt mask
 - DCMI_IT_OVF: Overflow interrupt mask
 - DCMI_IT_ERR: Synchronization error interrupt mask
 - DCMI_IT_VSYNC: VSYNC interrupt mask
 - DCMI_IT_LINE: Line interrupt mask

Return value:

- The state of INTERRUPT.

DCMI Extended Data Mode

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

DCMI Flags

DCMI_FLAG_HSYNC
DCMI_FLAG_VSYNC
DCMI_FLAG_FNE
DCMI_FLAG_FRAMERI
DCMI_FLAG_OVFRI
DCMI_FLAG_ERRRI
DCMI_FLAG_VSYNCRI
DCMI_FLAG_LINERI
DCMI_FLAG_FRAMEMI
DCMI_FLAG_OVFM
DCMI_FLAG_ERRM
DCMI_FLAG_VSYNCD
DCMI_FLAG_LINEMI

DCMI HSYNC Polarity

DCMI_HSPOLARITY_LOW Horizontal synchronization active Low
DCMI_HSPOLARITY_HIGH Horizontal synchronization active High

DCMI interrupt sources

DCMI_IT_FRAME
DCMI_IT_OVF
DCMI_IT_ERR
DCMI_IT_VSYNC
DCMI_IT_LINE

DCMI MODE JPEG

DCMI_JPEG_DISABLE Mode JPEG Disabled
DCMI_JPEG_ENABLE Mode JPEG Enabled

DCMI PIXCK Polarity

DCMI_PCKPOLARITY_FALLING Pixel clock active on Falling edge
DCMI_PCKPOLARITY_RISING Pixel clock active on Rising edge

DCMI Private Macros

IS_DCMI_CAPTURE_MODE
IS_DCMI_SYNCHRO
IS_DCMI_PCKPOLARITY
IS_DCMI_VSPOLARITY
IS_DCMI_HSPOLARITY
IS_DCMI_MODE_JPEG
IS_DCMI_CAPTURE_RATE

IS_DCMI_EXTENDED_DATA
IS_DCMI_WINDOW_COORDINATE
IS_DCMI_WINDOW_HEIGHT

DCMI Synchronization Mode

DCMI_SYNCHRO_HARDWARE	Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals
DCMI_SYNCHRO_EMBEDDED	Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

DCMI VSYNC Polarity

DCMI_VSPOLARITY_LOW	Vertical synchronization active Low
DCMI_VSPOLARITY_HIGH	Vertical synchronization active High

DCMI Window Coordinate

DCMI_WINDOW_COORDINATE Window coordinate

DCMI Window Height

DCMI_WINDOW_HEIGHT Window Height

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 ***DCMIE_Ex_Bit_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 ***DCMIE_Ex_Bit_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 ***DCMIE_Ex_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 ***DCMIE_Ex_Line_Select_Start***

16.2 DCMIE 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	HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef *hdcmi)
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"> hdcmi: pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
Return values	<ul style="list-style-type: none"> HAL status

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_HandleTypeDefDef::Instance`
DMA2D Register base address
- `DMA2D_InitTypeDef __DMA2D_HandleTypeDefDef::Init`
DMA2D communication parameters
- `void(* __DMA2D_HandleTypeDefDef::XferCpltCallback)(struct __DMA2D_HandleTypeDefDef *hdma2d)`
DMA2D transfer complete callback
- `void(* __DMA2D_HandleTypeDefDef::XferErrorCallback)(struct __DMA2D_HandleTypeDefDef *hdma2d)`
DMA2D transfer error callback
- `DMA2D_LayerCfgTypeDef __DMA2D_HandleTypeDefDef::LayerCfg[MAX_DMA2D_LAYER]`
DMA2D Layers parameters
- `HAL_LockTypeDef __DMA2D_HandleTypeDefDef::Lock`
DMA2D Lock
- `__IO HAL_DMA2D_StateTypeDef __DMA2D_HandleTypeDefDef::State`
DMA2D transfer state
- `__IO uint32_t __DMA2D_HandleTypeDefDef::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_DelInit\(\)**](#)
- [**HAL_DMA2D_MspInit\(\)**](#)
- [**HAL_DMA2D_MspDelInit\(\)**](#)

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

17.2.7 HAL_DMA2D_DelInit

Function Name	HAL_StatusTypeDef HAL_DMA2D_DelInit (DMA2D_HandleTypeDef * hdma2d)
Function Description	Deinitializes the DMA2D peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> • hdma2d: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	<ul style="list-style-type: none"> • None

17.2.8 HAL_DMA2D_MspInit

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

17.2.9 HAL_DMA2D_MspDelInit

Function Name	void HAL_DMA2D_MspDelInit (DMA2D_HandleTypeDef * hdma2d)
Function Description	Deinitializes the DMA2D MSP.
Parameters	<ul style="list-style-type: none"> • hdma2d: pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
Return values	<ul style="list-style-type: none"> • None

17.2.10 HAL_DMA2D_Start

Function Name	HAL_StatusTypeDef HAL_DMA2D_Start (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)
Function Description	Start the DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> • 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

17.2.11 HAL_DMA2D_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_DMA2D_Start_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)</code>
Function Description	Start the DMA2D Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

17.2.12 HAL_DMA2D_BlendingStart

Function Name	<code>HAL_StatusTypeDef HAL_DMA2D_BlendingStart (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)</code>
Function Description	Start the multi-source DMA2D Transfer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

17.2.13 HAL_DMA2D_BlendingStart_IT

Function Name	<code>HAL_StatusTypeDef HAL_DMA2D_BlendingStart_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width,</code>
---------------	---

uint32_t Height)

Function Description	Start the multi-source DMA2D Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

17.2.14 HAL_DMA2D_Abort

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

17.2.15 HAL_DMA2D_Suspend

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

17.2.16 HAL_DMA2D_Resume

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

17.2.17 HAL_DMA2D_PollForTransfer

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

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

17.2.18 HAL_DMA2D_IRQHandler

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

17.2.19 HAL_DMA2D_ConfigLayer

Function Name	HAL_StatusTypeDef HAL_DMA2D_ConfigLayer (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)
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"> 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	<ul style="list-style-type: none"> HAL status

17.2.20 HAL_DMA2D_ConfigCLUT

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

17.2.21 HAL_DMA2D_EnableCLUT

Function Name	HAL_StatusTypeDef HAL_DMA2D_EnableCLUT (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)
Function Description	Enable the DMA2D CLUT Transfer.
Parameters	<ul style="list-style-type: none"> 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

17.2.22 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 **Description:**

- Reset DMA2D handle state.

Parameters:

- _HANDLE_: specifies the DMA2D handle.

Return value:

- None

Description:

- Enable the DMA2D.

Parameters:

- `__HANDLE__`: DMA2D handle

Return value:

- None.

`__HAL_DMA2D_DISABLE`

Description:

- Disable the DMA2D.

Parameters:

- `__HANDLE__`: DMA2D handle

Return value:

- None.

`__HAL_DMA2D_GET_FLAG`

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.

`__HAL_DMA2D_CLEAR_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

- DMA2D_FLAG_TC: Transfer complete flag
- DMA2D_FLAG_TE: Transfer error flag

Return value:

- None

[__HAL_DMA2D_ENABLE_IT](#)**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

[__HAL_DMA2D_DISABLE_IT](#)**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

- DMA2D_IT_TE: Transfer error interrupt mask

Return value:

- None

`_HAL_DMA2D_GET_IT_SOURCE`**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.

DMA2D Exported Types`MAX_DMA2D_LAYER`**DMA2D Flag**

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

DMA2D Input Color Mode

<code>CM_ARGB8888</code>	ARGB8888 color mode
<code>CM_RGB888</code>	RGB888 color mode
<code>CM_RGB565</code>	RGB565 color mode
<code>CM_ARGB1555</code>	ARGB1555 color mode
<code>CM_ARGB4444</code>	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	HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)
Function Description	Initializes the DMA according to the specified parameters in the DMA_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hdma: Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
Return values	<ul style="list-style-type: none"> • HAL status

18.2.6 HAL_DMA_DeInit

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

18.2.7 HAL_DMA_Start

Function Name	HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)
Function Description	Starts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • 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 • DataLength: The length of data to be transferred from source to destination
Return values	<ul style="list-style-type: none"> • HAL status

18.2.8 HAL_DMA_Start_IT

Function Name	HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t
---------------	--

DstAddress, uint32_t DataLength)

Function Description	Start the DMA Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • 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 • DataLength: The length of data to be transferred from source to destination
Return values	<ul style="list-style-type: none"> • HAL status

18.2.9 HAL_DMA_Abort

Function Name	HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)
Function Description	Aborts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hdma: : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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.

18.2.10 HAL_DMA_PollForTransfer

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

18.2.11 HAL_DMA_IRQHandler

Function Name	void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)
Function Description	Handles DMA interrupt request.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
Return values	<ul style="list-style-type: none"> • None

18.2.12 HAL_DMA_GetState

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

18.2.13 HAL_DMA_GetError

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

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 requests

TIM_DMA_ID_CC2 Index of the DMA handle used for Capture/Compare 2 DMA requests

TIM_DMA_ID_CC3 Index of the DMA handle used for Capture/Compare 3 DMA requests

TIM_DMA_ID_CC4	Index of the DMA handle used for Capture/Compare 4 DMA requests
TIM_DMA_ID_COMMUTATION	Index of the DMA handle used for Commutation DMA requests
TIM_DMA_ID_TRIGGER	Index of the DMA handle used for Trigger DMA requests
DMA interrupt enable definitions	
DMA_IT_TC	
DMA_IT_HT	
DMA_IT_TE	
DMA_IT_DME	
DMA_IT_FE	
DMA Memory burst	
DMA_MBURST_SINGLE	
DMA_MBURST_INC4	
DMA_MBURST_INC8	
DMA_MBURST_INC16	
DMA Memory data size	
DMA_MDATAALIGN_BYTE	Memory data alignment: Byte
DMA_MDATAALIGN_HALFWORD	Memory data alignment: HalfWord
DMA_MDATAALIGN_WORD	Memory data alignment: Word
DMA Memory incremented mode	
DMA_MINC_ENABLE	Memory increment mode enable
DMA_MINC_DISABLE	Memory increment mode disable
DMA mode	
DMA_NORMAL	Normal mode
DMA_CIRCULAR	Circular mode
DMA_PFCTRL	Peripheral flow control mode
DMA Peripheral burst	
DMA_PBURST_SINGLE	
DMA_PBURST_INC4	
DMA_PBURST_INC8	
DMA_PBURST_INC16	
DMA Peripheral data size	
DMA_PDATAALIGN_BYTE	Peripheral data alignment: Byte
DMA_PDATAALIGN_HALFWORD	Peripheral data alignment: HalfWord
DMA_PDATAALIGN_WORD	Peripheral data alignment: Word
DMA Peripheral incremented mode	

DMA_PINC_ENABLE Peripheral increment mode enable
DMA_PINC_DISABLE Peripheral increment mode disable

DMA Priority level

DMA_PRIORITY_LOW Priority level: Low
DMA_PRIORITY_MEDIUM Priority level: Medium
DMA_PRIORITY_HIGH Priority level: High
DMA_PRIORITY VERY_HIGH Priority level: Very High

DMA Private Constants

HAL_TIMEOUT_DMA_ABORT

DMA Private Macros

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

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	<code>HAL_StatusTypeDef HAL_DMAEx_MultiBufferStart (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)</code>
Function Description	Starts the multi_buffer DMA Transfer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

19.1.4 HAL_DMAEx_MultiBufferStart_IT

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

DstAddress, uint32_t SecondMemAddress, uint32_t DataLength)

Function Description	Starts the multi_buffer DMA Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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 MedialInterface*

Field Documentation

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

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_Uncast_Frames_Filter`](#)
- **`uint32_t ETH_MACInitTypeDef::HashTableHigh`**
This field holds the higher 32 bits of Hash table. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFFFFFF
- **`uint32_t ETH_MACInitTypeDef::HashTableLow`**
This field holds the lower 32 bits of Hash table. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFFFFFF
- **`uint32_t ETH_MACInitTypeDef::PauseTime`**
This field holds the value to be used in the Pause Time field in the transmit control frame. This parameter must be a number between Min_Data = 0x0 and Max_Data = 0xFFFF
- **`uint32_t ETH_MACInitTypeDef::ZeroQuantaPause`**
Selects or not the automatic generation of Zero-Quanta Pause Control frames. This parameter can be a value of [`ETH_Zero_Quanta_Pause`](#)

- ***uint32_t ETH_MACInitTypeDef::PauseLowThreshold***
This field configures the threshold of the PAUSE to be checked for automatic retransmission of PAUSE Frame. This parameter can be a value of ***ETH_Pause_Low_Threshold***
- ***uint32_t ETH_MACInitTypeDef::UnicastPauseFrameDetect***
Selects or not the MAC detection of the Pause frames (with MAC Address0 unicast address and unique multicast address). This parameter can be a value of ***ETH_Uncast_Pause_Frame_Detect***
- ***uint32_t ETH_MACInitTypeDef::ReceiveFlowControl***
Enables or disables the MAC to decode the received Pause frame and disable its transmitter for a specified time (Pause Time) This parameter can be a value of ***ETH_Receive_Flow_Control***
- ***uint32_t ETH_MACInitTypeDef::TransmitFlowControl***
Enables or disables the MAC to transmit Pause frames (Full-Duplex mode) or the MAC back-pressure operation (Half-Duplex mode) This parameter can be a value of ***ETH_Transmit_Flow_Control***
- ***uint32_t ETH_MACInitTypeDef::VLANTagComparison***
Selects the 12-bit VLAN identifier or the complete 16-bit VLAN tag for comparison and filtering. This parameter can be a value of ***ETH_VLAN_Tag_Comparison***
- ***uint32_t ETH_MACInitTypeDef::VLANTagIdentifier***
Holds the VLAN tag identifier for receive frames

20.1.3 ETH_DMADef

Data Fields

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

Field Documentation

- ***uint32_t ETH_DMADef::DropTCPIPChecksumErrorFrame***
Selects or not the Dropping of TCP/IP Checksum Error Frames. This parameter can be a value of ***ETH_Drop_TCP_IP_Checksum_Error_Frame***
- ***uint32_t ETH_DMADef::ReceiveStoreForward***
Enables or disables the Receive store and forward mode. This parameter can be a value of ***ETH_Receive_Store_Forward***

- **`uint32_t ETH_DMADescTypeDef::Status`**
Enables or disables the flushing of received frames. This parameter can be a value of [`ETH_Flush_Received_Frame`](#)
- **`uint32_t ETH_DMADescTypeDef::ControlBufferSize`**
Enables or disables Transmit store and forward mode. This parameter can be a value of [`ETH_Transmit_Store_Forward`](#)
- **`uint32_t ETH_DMADescTypeDef::TransmitThresholdControl`**
Selects or not the Transmit Threshold Control. This parameter can be a value of [`ETH_Transmit_Threshold_Control`](#)
- **`uint32_t ETH_DMADescTypeDef::ForwardErrorFrames`**
Selects or not the forward to the DMA of erroneous frames. This parameter can be a value of [`ETH_Forward_Error_Frames`](#)
- **`uint32_t ETH_DMADescTypeDef::ForwardUndersizedGoodFrames`**
Enables or disables the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC) This parameter can be a value of [`ETH_Forward_Undersized_Good_Frames`](#)
- **`uint32_t ETH_DMADescTypeDef::ReceiveThresholdControl`**
Selects the threshold level of the Receive FIFO. This parameter can be a value of [`ETH_Receive_Threshold_Control`](#)
- **`uint32_t ETH_DMADescTypeDef::SecondFrameOperate`**
Selects or not the Operate on second frame mode, which allows the DMA to process a second frame of Transmit data even before obtaining the status for the first frame. This parameter can be a value of [`ETH_Second_Frame_Operate`](#)
- **`uint32_t ETH_DMADescTypeDef::AddressAlignedBeats`**
Enables or disables the Address Aligned Beats. This parameter can be a value of [`ETH_Address_Aligned_Beats`](#)
- **`uint32_t ETH_DMADescTypeDef::FixedBurst`**
Enables or disables the AHB Master interface fixed burst transfers. This parameter can be a value of [`ETH_Fixed_Burst`](#)
- **`uint32_t ETH_DMADescTypeDef::RxDMABurstLength`**
Indicates the maximum number of beats to be transferred in one Rx DMA transaction. This parameter can be a value of [`ETH_Rx_DMA_Burst_Length`](#)
- **`uint32_t ETH_DMADescTypeDef::TxDMABurstLength`**
Indicates the maximum number of beats to be transferred in one Tx DMA transaction. This parameter can be a value of [`ETH_Tx_DMA_Burst_Length`](#)
- **`uint32_t ETH_DMADescTypeDef::EnhancedDescriptorFormat`**
Enables the enhanced descriptor format. This parameter can be a value of [`ETH_DMA_Enhanced_descriptor_format`](#)
- **`uint32_t ETH_DMADescTypeDef::DescriptorSkipLength`**
Specifies the number of word to skip between two unchained descriptors (Ring mode) This parameter must be a number between Min_Data = 0 and Max_Data = 32
- **`uint32_t ETH_DMADescTypeDef::DMAArbitration`**
Selects the DMA Tx/Rx arbitration. This parameter can be a value of [`ETH_DMA_Arbitration`](#)

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_DMARxFrameInfos

Data Fields

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

Field Documentation

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

20.1.6 ETH_HandleTypeDef

Data Fields

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

Field Documentation

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

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_Delinit\(\)**](#)
- [**HAL_ETH_DMATxDescListInit\(\)**](#)
- [**HAL_ETH_DMARxDescListInit\(\)**](#)
- [**HAL_ETH_MspInit\(\)**](#)
- [**HAL_ETH_MspDelinit\(\)**](#)

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

20.2.7 HAL_ETH_DeInit

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

20.2.8 HAL_ETH_DMATxDescListInit

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

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

Return values

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"> • heth: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
	Return values	<ul style="list-style-type: none"> • HAL status
20.2.14 HAL_ETH_GetReceivedFrame_IT		
	Function Name	HAL_StatusTypeDef HAL_ETH_GetReceivedFrame_IT (ETH_HandleTypeDef * heth)
	Function Description	Gets the Received frame in interrupt mode.
	Parameters	<ul style="list-style-type: none"> • heth: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
	Return values	<ul style="list-style-type: none"> • HAL status
20.2.15 HAL_ETH_IRQHandler		
	Function Name	void HAL_ETH_IRQHandler (ETH_HandleTypeDef * heth)
	Function Description	This function handles ETH interrupt request.
	Parameters	<ul style="list-style-type: none"> • heth: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
	Return values	<ul style="list-style-type: none"> • HAL status
20.2.16 HAL_ETH_TxCpltCallback		
	Function Name	void HAL_ETH_TxCpltCallback (ETH_HandleTypeDef * heth)
	Function Description	Tx Transfer completed callbacks.
	Parameters	<ul style="list-style-type: none"> • heth: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
	Return values	<ul style="list-style-type: none"> • None
20.2.17 HAL_ETH_RxCpltCallback		
	Function Name	void HAL_ETH_RxCpltCallback (ETH_HandleTypeDef * heth)
	Function Description	Rx Transfer completed callbacks.
	Parameters	<ul style="list-style-type: none"> • heth: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
	Return values	<ul style="list-style-type: none"> • None
20.2.18 HAL_ETH_ErrorCallback		
	Function Name	void HAL_ETH_ErrorCallback (ETH_HandleTypeDef * heth)
	Function Description	Ethernet transfer error callbacks.
	Parameters	<ul style="list-style-type: none"> • heth: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module
	Return values	<ul style="list-style-type: none"> • None

20.2.19 HAL_ETH_ReadPHYRegister

Function Name	<code>HAL_StatusTypeDef HAL_ETH_ReadPHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t * RegValue)</code>
Function Description	Reads a PHY register.
Parameters	<ul style="list-style-type: none"> • heth: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module • PHYReg: PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY_BCR: Transceiver Basic Control Register, PHY_BSR: Transceiver Basic Status Register. More PHY register could be read depending on the used PHY • RegValue: PHY register value
Return values	<ul style="list-style-type: none"> • HAL status

20.2.20 HAL_ETH_WritePHYRegister

Function Name	<code>HAL_StatusTypeDef HAL_ETH_WritePHYRegister (ETH_HandleTypeDef * heth, uint16_t PHYReg, uint32_t RegValue)</code>
Function Description	Writes to a PHY register.
Parameters	<ul style="list-style-type: none"> • heth: pointer to a ETH_HandleTypeDef structure that contains the configuration information for ETHERNET module • PHYReg: PHY register address, is the index of one of the 32 PHY register. This parameter can be one of the following values: PHY_BCR: Transceiver Control Register. More PHY register could be written depending on the used PHY • RegValue: the value to write
Return values	<ul style="list-style-type: none"> • HAL status

20.2.21 HAL_ETH_Start

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

20.2.22 HAL_ETH_Stop

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

20.2.23 HAL_ETH_ConfigMAC

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

20.2.24 HAL_ETH_ConfigDMA

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

20.2.25 HAL_ETH_GetState

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

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_RXBUFN

ETH_TX_BUF_SIZE

ETH_TXBUFN

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_DEFERRALCHECK_ENABLE

ETH_DEFERRALCHECK_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_QUEUEING	Running - queuing the receive frame into host memory

ETH DMA RX Descriptor

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

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

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

ETH_DMADMAPTRXDESC_RTSH

ETH DMA Rx descriptor buffers

ETH_DMARXDESC_BUFFER1 DMA Rx Desc Buffer1

ETH_DMARXDESC_BUFFER2 DMA Rx Desc Buffer2

ETH DMA transmit process state

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_DMAPPTXDESC_TTSL	
ETH_DMAPPTXDESC_TTSH	

ETH DMA Tx descriptor Checksum Insertion Control

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

ETH DMA Tx descriptor segment

ETH_DMATXDESC_LASTSEGMENTS	Last Segment
ETH_DMATXDESC_FIRSTSEGMENT	First Segment

ETH Drop TCP IP Checksum Error Frame

ETH_DROPTCPIPCHECKSUMERRORFRAME_ENABLE

ETH_DROPTCIPCHECKSUMERRORFRAME_DISABLE

ETH Duplex Mode

ETH_MODE_FULLDUPLEX

ETH_MODE_HALFDUPLEX

ETH Exported Macros

`__HAL_ETH_RESET_HANDLE_STATE`

Description:

- Reset ETH handle state.

Parameters:

- `__HANDLE__`: specifies the ETH handle.

Return value:

- None

`__HAL_ETH_DMATXDESC_GET_FLAG`

Description:

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

Parameters:

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

Return value:

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

`__HAL_ETH_DMARXDESC_GET_FLAG`

Description:

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

Parameters:

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

Return value:

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

`__HAL_ETH_DMARXDESC_ENABLE_IT`

Description:

- Enables the specified DMA Rx Desc receive interrupt.

Parameters:

- `__HANDLE__`: ETH Handle

Return value:

- None

__HAL_ETH_DMARXDESC_DISABLE_IT

Description:

- Disables the specified DMA Rx Desc receive interrupt.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None

__HAL_ETH_DMARXDESC_SET_OWN_BIT

Description:

- Set the specified DMA Rx Desc Own bit.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None

__HAL_ETH_DMATXDESC_GET_COLLISION_COUNT

Description:

- Returns the specified ETHERNET DMA Tx Desc collision count.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- The: Transmit descriptor collision counter value.

__HAL_ETH_DMATXDESC_SET_OWN_BIT

Description:

- Set the specified DMA Tx Desc Own bit.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None

__HAL_ETH_DMATXDESC_ENABLE_IT

Description:

- Enables the specified DMA Tx Desc Transmit interrupt.

Parameters:

- __HANDLE__: ETH Handle

Return value:

- None

__HAL_ETH_DMATXDESC_DISABLE_IT

Description:

- Disables the specified DMA Tx Desc

Transmit interrupt.

Parameters:

- `__HANDLE__`: ETH Handle

Return value:

- None

`_HAL_ETH_DMATXDESC_CHECKSUM
_INSERTION`

Description:

- Selects the specified ETHERNET DMA Tx Desc Checksum Insertion.

Parameters:

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

Return value:

- None

`_HAL_ETH_DMATXDESC_CRC_ENABLE`

Description:

- Enables the DMA Tx Desc CRC.

Parameters:

- `__HANDLE__`: ETH Handle

Return value:

- None

`_HAL_ETH_DMATXDESC_CRC_DISABLE`

Description:

- Disables the DMA Tx Desc CRC.

Parameters:

- `__HANDLE__`: ETH Handle

Return value:

- None

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

- None

`_HAL_ETH_INITIATE_PAUSE_CONTROL_FRAME`

Description:

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

Parameters:

- `_HANDLE_`: ETH Handle

Return value:

- None

`_HAL_ETH_GET_FLOW_CONTROL_BUSY_STATUS`

Description:

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

Parameters:

- `_HANDLE_`: ETH Handle

Return value:

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

`_HAL_ETH_BACK_PRESSURE_ACTIVATION_ENABLE`

Description:

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

Parameters:

- `_HANDLE_`: ETH Handle

Return value:

- None

`_HAL_ETH_BACK_PRESSURE_ACTIVATION_DISABLE`

Description:

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

Parameters:

- `_HANDLE_`: ETH Handle

Return value:

- None

`_HAL_ETH_MAC_GET_FLAG`

Description:

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

Parameters:

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

- ETH_MAC_FLAG_MMCT : MMC transmit flag
- ETH_MAC_FLAG_MMCR : MMC receive flag
- ETH_MAC_FLAG_MMIC : MMC flag
- ETH_MAC_FLAG_PMT : PMT flag

Return value:

- The state of ETHERNET MAC flag.

[__HAL_ETH_DMA_ENABLE_IT](#)

Description:

- Enables the specified ETHERNET DMA interrupts.

Parameters:

- [__HANDLE__](#): ETH Handle
- [__INTERRUPT__](#): specifies the ETHERNET DMA interrupt sources to be enabled

Return value:

- None

[__HAL_ETH_DMA_DISABLE_IT](#)

Description:

- Disables the specified ETHERNET DMA interrupts.

Parameters:

- [__HANDLE__](#): ETH Handle
- [__INTERRUPT__](#): specifies the ETHERNET DMA interrupt sources to be disabled.

Return value:

- None

[__HAL_ETH_DMA_CLEAR_IT](#)

Description:

- Clears the ETHERNET DMA IT pending bit.

Parameters:

- [__HANDLE__](#): ETH Handle
- [__INTERRUPT__](#): specifies the interrupt pending bit to clear.

Return value:

- None

[__HAL_ETH_DMA_GET_FLAG](#)

Description:

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

Parameters:

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

Return value:

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

[__HAL_ETH_DMA_CLEAR_FLAG](#)**Description:**

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

Parameters:

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

Return value:

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

[__HAL_ETH_GET_DMA_OVERFLOW_STATUS](#)**Description:**

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

Parameters:

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

Return value:

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

[__HAL_ETH_SET_RECEIVE_WATCHDOG_TIMER](#)**Description:**

- Set the DMA Receive status watchdog timer register value.

Parameters:

- __HANDLE__: ETH Handle
- __VALUE__: DMA Receive status watchdog timer register value

Return value:

- None

[__HAL_ETH_GLOBAL_UNICAST_WAKE](#)**Description:**

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

Description:

- Preset and Initialize the MMC counters to almost-full value: 0xFFFF_FFF0 (full - 16)

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

`__HAL_ETH_MMC_COUNTER_HALF_PR
ESET`

Description:

- Preset and Initialize the MMC counters to almost-half value: 0x7FFF_FFF0 (half - 16)

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

`__HAL_ETH_MMC_COUNTER_FREEZE_
ENABLE`

Description:

- Enables the MMC Counter Freeze.

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

`__HAL_ETH_MMC_COUNTER_FREEZE_
DISABLE`

Description:

- Disables the MMC Counter Freeze.

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

`__HAL_ETH_ETH_MMC_RESET_ONREA
D_ENABLE`

Description:

- Enables the MMC Reset On Read.

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

`__HAL_ETH_ETH_MMC_RESET_ONREA
D_DISABLE`

Description:

- Disables the MMC Reset On Read.

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

`__HAL_ETH_ETH_MMC_COUNTER_RO
LLOVER_ENABLE`

Description:

- Enables the MMC Counter Stop

Rollover.

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

Description:

- Disables the MMC Counter Stop Rollover.

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

Description:

- Resets the MMC Counters.

Parameters:

- `__HANDLE__`: ETH Handle.

Return value:

- None

Description:

- Enables the specified ETHERNET MMC Rx interrupts.

Parameters:

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

Return value:

- None

Description:

- Disables the specified ETHERNET MMC Rx interrupts.

Parameters:

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

Return value:

- None

[__HAL_ETH_MMC_TX_IT_ENABLE](#)**Description:**

- Enables the specified ETHERNET MMC Tx interrupts.

Parameters:

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

Return value:

- None

[__HAL_ETH_MMC_TX_IT_DISABLE](#)**Description:**

- Disables the specified ETHERNET MMC Tx interrupts.

Parameters:

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

- the maximum value
- ETH_MMC_IT_TGFMSC: When Tx good multi col counter reaches half the maximum value
- ETH_MMC_IT_TGFSC : When Tx good single col counter reaches half the maximum value

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_IT`

Description:

- Enables the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_IT`

Description:

- Disables the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_EVENT`

Description:

- Enable event on ETH External event line.

Return value:

- None.

`__HAL_ETH_WAKEUP_EXTI_DISABLE_EVENT`

Description:

- Disable event on ETH External event line.

Return value:

- None.

`__HAL_ETH_WAKEUP_EXTI_GET_FLAG`

Description:

- Get flag of the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_CLEAR_FLAG`

Description:

- Clear flag of the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_RISING_EDGE_TRIGGER`

Description:

- Enables rising edge trigger to the ETH

External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_
RISING_EDGE_TRIGGER`

Description:

- Disables the rising edge trigger to the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_
FALLING_EDGE_TRIGGER`

Description:

- Enables falling edge trigger to the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_
FALLING_EDGE_TRIGGER`

Description:

- Disables falling edge trigger to the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_ENABLE_
FALLINGRISING_TRIGGER`

Description:

- Enables rising/falling edge trigger to the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_DISABLE_
FALLINGRISING_TRIGGER`

Description:

- Disables rising/falling edge trigger to the ETH External interrupt line.

Return value:

- None

`__HAL_ETH_WAKEUP_EXTI_GENERAT
E_SWIT`

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None.

ETH EXTI LINE WAKEUP

`ETH_EXTI_LINE_WAKEUP` External interrupt line 19 Connected to the ETH EXTI Line

ETH Fixed Burst

`ETH_FIXEDBURST_ENABLE`

`ETH_FIXEDBURST_DISABLE`

ETH Flush Received Frame

ETH_FLUSHRECEIVEDFRAME_ENABLE

ETH_FLUSHRECEIVEDFRAME_DISABLE

ETH Forward Error Frames

ETH_FORWARDERRORFRAMES_ENABLE

ETH_FORWARDERRORFRAMES_DISABLE

ETH Forward Undersized Good Frames

ETH_FORWARDUNDERSIZEDGOODFRAMES_ENABLE

ETH_FORWARDUNDERSIZEDGOODFRAMES_DISABLE

ETH Inter Frame Gap

ETH_INTERFRAMEGAP_96BIT minimum IFG between frames during transmission is 96Bit

ETH_INTERFRAMEGAP_88BIT minimum IFG between frames during transmission is 88Bit

ETH_INTERFRAMEGAP_80BIT minimum IFG between frames during transmission is 80Bit

ETH_INTERFRAMEGAP_72BIT minimum IFG between frames during transmission is 72Bit

ETH_INTERFRAMEGAP_64BIT minimum IFG between frames during transmission is 64Bit

ETH_INTERFRAMEGAP_56BIT minimum IFG between frames during transmission is 56Bit

ETH_INTERFRAMEGAP_48BIT minimum IFG between frames during transmission is 48Bit

ETH_INTERFRAMEGAP_40BIT minimum IFG between frames during transmission is 40Bit

ETH Jabber

ETH_JABBER_ENABLE

ETH_JABBER_DISABLE

ETH Loop Back Mode

ETH_LOOPBACKMODE_ENABLE

ETH_LOOPBACKMODE_DISABLE

ETH MAC addresses

ETH_MAC_ADDRESS0

ETH_MAC_ADDRESS1

ETH_MAC_ADDRESS2

ETH_MAC_ADDRESS3

ETH MAC addresses filter Mask bytes

ETH_MAC_ADDRESSMASK_BYTE6 Mask MAC Address high reg bits [15:8]

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

ETH MAC addresses filter SA DA

ETH_MAC_ADDRESSFILTER_SA
ETH_MAC_ADDRESSFILTER_DA

ETH MAC Debug flags

ETH_MAC_TXFIFO_FULL
ETH_MAC_TXFIFONOT_EMPTY
ETH_MAC_TXFIFO_WRITE_ACTIVE
ETH_MAC_TXFIFO_IDLE
ETH_MAC_TXFIFO_READ
ETH_MAC_TXFIFO_WAITING
ETH_MAC_TXFIFO_WRITING
ETH_MAC_TRANSMISSION_PAUSE
ETH_MAC_TRANSMITFRAMECONTROLLER_IDLE
ETH_MAC_TRANSMITFRAMECONTROLLER_WAITING
ETH_MAC_TRANSMITFRAMECONTROLLER_GENRATING_PCF
ETH_MAC_TRANSMITFRAMECONTROLLER_TRANSFERRING
ETH_MAC_MII_TRANSMIT_ACTIVE
ETH_MAC_RXFIFO_EMPTY
ETH_MAC_RXFIFO_BELOW_THRESHOLD
ETH_MAC_RXFIFO_ABOVE_THRESHOLD
ETH_MAC_RXFIFO_FULL
ETH_MAC_READCONTROLLER_IDLE
ETH_MAC_READCONTROLLER_READING_DATA
ETH_MAC_READCONTROLLER_READING_STATUS
ETH_MAC_READCONTROLLER_
ETH_MAC_RXFIFO_WRITE_ACTIVE
ETH_MAC_SMALL_FIFO_NOTACTIVE
ETH_MAC_SMALL_FIFO_READ_ACTIVE
ETH_MAC_SMALL_FIFO_WRITE_ACTIVE
ETH_MAC_SMALL_FIFO_RW_ACTIVE
ETH_MAC_MII_RECEIVE_PROTOCOL_ACTIVE

ETH MAC Flags

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

ETH MAC Interrupts

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

ETH Media Interface

ETH_MEDIA_INTERFACE_MII	
ETH_MEDIA_INTERFACE_RMII	

ETH MMC Rx Interrupts

ETH_MMCI_T_RGUF	When Rx good unicast frames counter reaches half the maximum value
ETH_MMCI_T_RFAE	When Rx alignment error counter reaches half the maximum value
ETH_MMCI_T_RFCE	When Rx crc error counter reaches half the maximum value

ETH MMC Tx Interrupts

ETH_MMCI_T_TGF	When Tx good frame counter reaches half the maximum value
ETH_MMCI_T_TGFMSC	When Tx good multi col counter reaches half the maximum value
ETH_MMCI_T_TGFSC	When Tx good single col counter reaches half the maximum value

ETH Multicast Frames Filter

ETH_MULTICASTFRAMESFILTER_PERFECTHASHTABLE	
ETH_MULTICASTFRAMESFILTER_HASHTABLE	
ETH_MULTICASTFRAMESFILTER_PERFECT	
ETH_MULTICASTFRAMESFILTER_NONE	

ETH Pass Control Frames

ETH_PASSCONTROLFRAMES_BLOCKALL	MAC filters all control frames from reaching the application
ETH_PASSCONTROLFRAMES_FORWARDALL	MAC forwards all control frames to application even if they fail the Address Filter

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_WUFRPR	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_SIZESSHIFT
ETH_DMARXDESC_FRAME_LENGTHSHIFT
ETH_DMARXDESC_BUFFER2_SIZESSHIFT
ETH_DMARXDESC_FRAMELENGTHSHIFT
ETH_MAC_ADDR_HBASE
ETH_MAC_ADDR_LBASE
ETH_MACIIAR_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	<code>HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t Data)</code>
Function Description	Program byte, halfword, word or double word at a specified address.
Parameters	<ul style="list-style-type: none"> • TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • Address: specifies the address to be programmed. • Data: specifies the data to be programmed
Return values	• <code>HAL_StatusTypeDef HAL Status</code>

21.2.7 HAL_FLASH_Program_IT

Function Name	<code>HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)</code>
Function Description	Program byte, halfword, word or double word at a specified address with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • Address: specifies the address to be programmed. • Data: specifies the data to be programmed
Return values	• <code>HAL Status</code>

21.2.8 HAL_FLASH_IRQHandler

Function Name	<code>void HAL_FLASH_IRQHandler (void)</code>
Function Description	This function handles FLASH interrupt request.
Return values	<ul style="list-style-type: none"> • None

21.2.9 HAL_FLASH_EndOfOperationCallback

Function Name	<code>void HAL_FLASH_EndOfOperationCallback (uint32_t</code>
---------------	--

ReturnValue

Function Description	FLASH end of operation interrupt callback.
Parameters	<ul style="list-style-type: none"> • ReturnValue: The value saved in this parameter depends on the ongoing procedureSectors 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 programMass Erase : No return value expected
Return values	<ul style="list-style-type: none"> • None

21.2.10 HAL_FLASH_OperationErrorCallback

Function Name	void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)
Function Description	FLASH operation error interrupt callback.
Parameters	<ul style="list-style-type: none"> • ReturnValue: The value saved in this parameter depends on the ongoing procedureSectors 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 programMass Erase : No return value expected
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL Status

21.2.15 HAL_FLASH_OB_Launch

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)
Function Description	Launch the option byte loading.
Return values	<ul style="list-style-type: none"> • HAL Status

21.2.16 HAL_FLASH_GetError

Function Name	uint32_t HAL_FLASH_GetError (void)
Function Description	Get the specific FLASH error flag.
Return values	<ul style="list-style-type: none"> • 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

21.2.17 FLASH_WaitForLastOperation

Function Name	HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t Timeout)
Function Description	Wait for a FLASH operation to complete.
Parameters	<ul style="list-style-type: none"> • Timeout: maximum flash operationtimeout
Return values	<ul style="list-style-type: none"> • HAL Status

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

<code>_HAL_FLASH_SET_LATENCY</code>	Description: <ul style="list-style-type: none"> • Set the FLASH Latency. Parameters: <ul style="list-style-type: none"> • <code>_LATENCY_</code>: FLASH Latency The value of this parameter depend on device used within the same series Return value:
-------------------------------------	---

- none

Description:

- Enable the FLASH prefetch buffer.

Return value:

- none

Description:

- Disable the FLASH prefetch buffer.

Return value:

- none

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.

Description:

- Disable the FLASH Adaptive Real-Time memory accelerator.

Return value:

- none

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.

Description:

- Enable the specified FLASH interrupt.

Parameters:

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

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

Return value:

- none

[__HAL_FLASH_DISABLE_IT](#)**Description:**

- Disable the specified FLASH interrupt.

Parameters:

- [__INTERRUPT__](#): : FLASH interrupt This parameter can be any combination of the following values:
 - FLASH_IT_EOP: End of FLASH Operation Interrupt
 - FLASH_IT_ERR: Error Interrupt

Return value:

- none

[__HAL_FLASH_GET_FLAG](#)**Description:**

- Get the specified FLASH flag status.

Parameters:

- [__FLAG__](#): specifies the FLASH flag to check. This parameter can be one of the following values:
 - FLASH_FLAG_EOP : FLASH End of Operation flag
 - FLASH_FLAG_OPERR : FLASH operation Error flag
 - FLASH_FLAG_WRPERR: FLASH Write protected error flag
 - FLASH_FLAG_PGAERR: FLASH Programming Alignment error flag
 - FLASH_FLAG_PGPERR: FLASH Programming Parallelism error flag
 - FLASH_FLAG_ERSERR : FLASH Erasing Sequence error flag
 - FLASH_FLAG_BSY : FLASH Busy flag

Return value:

- The: new state of [__FLAG__](#) (SET

or RESET).

`_HAL_FLASH_CLEAR_FLAG`

Description:

- Clear the specified FLASH flag.

Parameters:

- `_FLAG_`: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
 - `FLASH_FLAG_EOP` : FLASH End of Operation flag
 - `FLASH_FLAG_OPERR` : FLASH operation Error flag
 - `FLASH_FLAG_WRPERR` : FLASH Write protected error flag
 - `FLASH_FLAG_PGAERR` : FLASH Programming Alignment error flag
 - `FLASH_FLAG_PGPERR` : FLASH Programming Parallelism error flag
 - `FLASH_FLAG_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 RDPLevel*
- *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::RDPLevel***
Set the read protection level. This parameter can be a value of ***FLASHEx_Option_Bytes_Read_Protection***
- ***uint32_t FLASH_OBProgramInitTypeDef::BORLevel***
Set the BOR Level. This parameter can be a value of ***FLASHEx_BOR_Reset_Level***
- ***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 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	HAL_StatusTypeDef HAL_FLASHEx_Erase (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">• pEraseInit: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.• SectorError: 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)
Return values	<ul style="list-style-type: none">• HAL Status

22.2.5 HAL_FLASHEx_Erase_IT

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)
Function Description	Perform a mass erase or erase the specified FLASH memory sectors with interrupt enabled.
Parameters	<ul style="list-style-type: none">• pEraseInit: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
Return values	<ul style="list-style-type: none">• HAL Status

22.2.6 HAL_FLASHEx_OBProgram

Function Name	HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none">• pOBInit: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none">• HAL Status

22.2.7 HAL_FLASHEx_OBGetConfig

Function Name	void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)
Function Description	Get the Option byte configuration.
Parameters	<ul style="list-style-type: none"> pOBInit: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> None

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	<code>void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct)</code>
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the <code>GPIO_Init</code> .
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..K) to select the GPIO peripheral. • GPIO_InitStruct: pointer to a <code>GPIO_InitTypeDef</code> structure that contains the configuration information for the specified GPIO peripheral.
Return values	<ul style="list-style-type: none"> • None

23.2.6 `HAL_GPIO_DeInit`

Function Name	<code>void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)</code>
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..K) to select the GPIO peripheral. • GPIO_Pin: specifies the port bit to be written. This parameter can be one of <code>GPIO_PIN_x</code> where x can be (0..15).
Return values	<ul style="list-style-type: none"> • None

23.2.7 `HAL_GPIO_ReadPin`

Function Name	<code>GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Reads the specified input port pin.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..K) to select the GPIO peripheral. • GPIO_Pin: specifies the port bit to read. This parameter can be <code>GPIO_PIN_x</code> where x can be (0..15).
Return values	<ul style="list-style-type: none"> • The input port pin value.

23.2.8 `HAL_GPIO_WritePin`

Function Name	<code>void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)</code>
Function Description	Sets or clears the selected data port bit.

Parameters	<ul style="list-style-type: none"> GPIOx: where x can be (A..K) to select the GPIO peripheral. GPIO_Pin: specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). PinState: specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_PIN_RESET: to clear the port pinGPIO_PIN_SET: to set the port pin
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> 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.

23.2.9 HAL_GPIO_TogglePin

Function Name	<code>void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Toggles the specified GPIO pins.
Parameters	<ul style="list-style-type: none"> GPIOx: Where x can be (A..I) to select the GPIO peripheral. GPIO_Pin: Specifies the pins to be toggled.
Return values	<ul style="list-style-type: none"> None

23.2.10 HAL_GPIO_LockPin

Function Name	<code>HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)</code>
Function Description	Locks GPIO Pins configuration registers.
Parameters	<ul style="list-style-type: none"> GPIOx: where x can be (A..F) to select the GPIO peripheral for STM32F7 family GPIO_Pin: specifies the port bit to be locked. This parameter can be any combination of GPIO_PIN_x where x can be (0..15).
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH. The configuration of the locked GPIO pins can no longer be modified until the next reset.

23.2.11 HAL_GPIO_EXTI_IRQHandler

Function Name	<code>void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)</code>
Function Description	This function handles EXTI interrupt request.
Parameters	<ul style="list-style-type: none"> GPIO_Pin: Specifies the pins connected EXTI line
Return values	<ul style="list-style-type: none"> None

23.2.12 HAL_GPIO_EXTI_Callback

Function Name	void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)
Function Description	EXTI line detection callbacks.
Parameters	<ul style="list-style-type: none">• GPIO_Pin: Specifies the pins connected EXTI line
Return values	<ul style="list-style-type: none">• None

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:

- `__EXTI_LINE__`: specifies the EXTI line flag to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

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

`_HAL_GPIO_EXTI_CLEAR_FLAG`

Description:

- Clears the EXTI's line pending flags.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI lines flags to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

Return value:

- None

`_HAL_GPIO_EXTI_GET_IT`

Description:

- Checks whether the specified EXTI line is asserted or not.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

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

`_HAL_GPIO_EXTI_CLEAR_IT`

Description:

- Clears the EXTI's line pending bits.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI lines to clear. This parameter can be any combination of `GPIO_PIN_x` where x can be (0..15)

Return value:

- None

`_HAL_GPIO_EXTI_GENERATE_SWIT`

Description:

- Generates a Software interrupt on selected EXTI line.

Parameters:

- `__EXTI_LINE__`: specifies the EXTI line to check. This parameter can be `GPIO_PIN_x` where x can be(0..15)

Return value:

- None

GPIO mode define

GPIO_MODE_INPUT	Input Floating Mode
GPIO_MODE_OUTPUT_PP	Output Push Pull Mode
GPIO_MODE_OUTPUT_OD	Output Open Drain Mode
GPIO_MODE_AF_PP	Alternate Function Push Pull Mode
GPIO_MODE_AF_OD	Alternate Function Open Drain Mode
GPIO_MODE_ANALOG	Analog Mode
GPIO_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
GPIO_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
GPIO_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
GPIO_MODE_EVT_RISING	External Event Mode with Rising edge trigger detection
GPIO_MODE_EVT_FALLING	External Event Mode with Falling edge trigger detection
GPIO_MODE_EVT_RISING_FALLING	External Event Mode with Rising/Falling edge trigger detection

GPIO pins define

GPIO_PIN_0
GPIO_PIN_1
GPIO_PIN_2
GPIO_PIN_3
GPIO_PIN_4
GPIO_PIN_5
GPIO_PIN_6
GPIO_PIN_7
GPIO_PIN_8
GPIO_PIN_9
GPIO_PIN_10
GPIO_PIN_11
GPIO_PIN_12
GPIO_PIN_13
GPIO_PIN_14
GPIO_PIN_15

GPIO_PIN_All

GPIO_PIN_MASK

GPIO Private Constants

GPIO_MODE

EXTI_MODE

GPIO_MODE_IT

GPIO_MODE_EVT

RISING_EDGE

FALLING_EDGE

GPIO_OUTPUT_TYPE

GPIO_NUMBER

GPIO Private Macros

IS_GPIO_PIN_ACTION

IS_GPIO_PIN

IS_GPIO_MODE

IS_GPIO_SPEED

IS_GPIO_PULL

GPIO pull define

GPIO_NOPULL No Pull-up or Pull-down activation

GPIO_PULLUP Pull-up activation

GPIO_PULLDOWN Pull-down activation

GPIO speed define

GPIO_SPEED_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_STATETypeDef State*

Field Documentation

- ***HASH_InitTypeDef HASH_HandleTypeDef::Init***
HASH required parameters
- ***uint8_t* HASH_HandleTypeDef::pHashInBuffPtr***
Pointer to input buffer
- ***uint8_t* HASH_HandleTypeDef::pHashOutBuffPtr***
Pointer to output buffer

- **`_IO uint32_t HASH_HandleTypeDef::HashBuffSize`**
Size of buffer to be processed
- **`_IO uint32_t HASH_HandleTypeDef::HashInCount`**
Counter of inputed data
- **`_IO uint32_t HASH_HandleTypeDef::HashITCounter`**
Counter of issued interrupts
- **`HAL_StatusTypeDef HASH_HandleTypeDef::Status`**
HASH peripheral status
- **`HAL_HASHPhaseTypeDef HASH_HandleTypeDef::Phase`**
HASH peripheral phase
- **`DMA_HandleTypeDef* HASH_HandleTypeDef::hdmain`**
HASH In DMA handle parameters
- **`HAL_LockTypeDef HASH_HandleTypeDef::Lock`**
HASH locking object
- **`_IO HAL_HASH_STATETypeDef HASH_HandleTypeDef::State`**
HASH peripheral state

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	HAL_StatusTypeDef HAL_HASH_MD5_Start (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"> • 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 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. • pOutBuffer: Pointer to the computed digest. Its size must be 16 bytes. • Timeout: Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

25.2.10 HAL_HASH_MD5_Accumulate

Function Name	HAL_StatusTypeDef HAL_HASH_MD5_Accumulate (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"> • 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 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.
Return values	<ul style="list-style-type: none"> • HAL status

25.2.11 HAL_HASH_SHA1_Start

Function Name	HAL_StatusTypeDef HAL_HASH_SHA1_Start (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"> • 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.

- **Timeout:** Timeout value
- HAL status

25.2.12 HAL_HASH_SHA1_Accumulate

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

25.2.13 HAL_HASH_MD5_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_HASH_MD5_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)</code>
Function Description	Initializes the HASH peripheral in MD5 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

25.2.14 HAL_HASH_SHA1_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)</code>
Function Description	Initializes the HASH peripheral in SHA1 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

25.2.15 HAL_HASH_IRQHandler

Function Name	void HAL_HASH_IRQHandler (HASH_HandleTypeDef * hhash)
Function Description	This function handles HASH interrupt request.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.16 HAL_HMAC_SHA1_Start

Function Name	HAL_StatusTypeDef HAL_HMAC_SHA1_Start (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"> • 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. • Timeout: Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

25.2.17 HAL_HMAC_MD5_Start

Function Name	HAL_StatusTypeDef HAL_HMAC_MD5_Start (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"> • 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. • Timeout: Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

25.2.18 HAL_HASH_MD5_Start_DMA

Function Name	HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA (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"> 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	<ul style="list-style-type: none"> HAL status

25.2.19 HAL_HASH_MD5_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASH_MD5_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Returns the computed digest in MD5 mode.
Parameters	<ul style="list-style-type: none"> hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module pOutBuffer: Pointer to the computed digest. Its size must be 16 bytes. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

25.2.20 HAL_HASH_SHA1_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in SHA1 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

25.2.21 HAL_HASH_SHA1_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASH_SHA1_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Returns the computed digest in SHA1 mode.
Parameters	<ul style="list-style-type: none"> hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module pOutBuffer: Pointer to the computed digest. Its size must be 20 bytes. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

25.2.22 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

25.2.23 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

25.2.24 HAL_HMAC_MD5_Start

Function Name	HAL_StatusTypeDef HAL_HMAC_MD5_Start (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"> 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. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

25.2.25 HAL_HMAC_SHA1_Start

Function Name	HAL_StatusTypeDef HAL_HMAC_SHA1_Start (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"> 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. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

25.2.26 HAL_HASH_SHA1_Start_DMA

Function Name	HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA (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"> 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	<ul style="list-style-type: none"> HAL status

25.2.27 HAL_HASH_SHA1_Finish

Function Name	HAL_StatusTypeDef HAL_HASH_SHA1_Finish (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"> hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module pOutBuffer: Pointer to the computed digest. Its size must be 20 bytes. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

25.2.28 HAL_HASH_MD5_Start_DMA

Function Name	HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA (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"> 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	<ul style="list-style-type: none"> HAL status

25.2.29 HAL_HASH_MD5_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASH_MD5_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Returns the computed digest in MD5 mode.
Parameters	<ul style="list-style-type: none"> hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module pOutBuffer: Pointer to the computed digest. Its size must be 16 bytes. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

25.2.30 HAL_HMAC_MD5_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HMAC_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in HMAC MD5 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

25.2.31 HAL_HMAC_SHA1_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HMAC_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA1 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

25.2.32 HAL_HASH_GetState

Function Name	HAL_HASH_STATETypeDef HAL_HASH_GetState (HASH_HandleTypeDef * hhash)
Function Description	return the HASH state
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • HAL state

25.2.33 HAL_HASH_IRQHandler

Function Name	void HAL_HASH_IRQHandler (HASH_HandleTypeDef * hhash)
Function Description	This function handles HASH interrupt request.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.34 HAL_HASH_Init

Function Name	HAL_StatusTypeDef HAL_HASH_Init (HASH_HandleTypeDef * hhash)
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"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • HAL status

25.2.35 HAL_HASH_DeInit

Function Name	HAL_StatusTypeDef HAL_HASH_DeInit (HASH_HandleTypeDef * hhash)
Function Description	Deinitializes the HASH peripheral.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- This API must be called before starting a new processing.

25.2.36 HAL_HASH_MspInit

Function Name	void HAL_HASH_MspInit (HASH_HandleTypeDef * hhash)
Function Description	Initializes the HASH MSP.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.37 HAL_HASH_MspDeInit

Function Name	void HAL_HASH_MspDelnit (HASH_HandleTypeDef * hhash)
Function Description	Deinitializes HASH MSP.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.38 HAL_HASH_InCpltCallback

Function Name	void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)
Function Description	Input data transfer complete callback.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.39 HAL_HASH_ErrorCallback

Function Name	void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)
Function Description	Data transfer Error callback.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.40 HAL_HASH_DgstCpltCallback

Function Name	void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)
Function Description	Digest computation complete callback.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

Notes

- This callback is not relevant with DMA.

25.2.41 HAL_HASH_GetState

Function Name	HAL_HASH_STATETypeDef HAL_HASH_GetState (HASH_HandleTypeDef * hhash)
Function Description	return the HASH state
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • HAL state

25.2.42 HAL_HASH_MspInit

Function Name	void HAL_HASH_MspInit (HASH_HandleTypeDef * hhash)
Function Description	Initializes the HASH MSP.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.43 HAL_HASH_MspDeInit

Function Name	void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)
Function Description	Deinitializes HASH MSP.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.44 HAL_HASH_InCpltCallback

Function Name	void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)
Function Description	Input data transfer complete callback.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

25.2.45 HAL_HASH_DgstCpltCallback

Function Name	void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)
Function Description	Digest computation complete callback.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This callback is not relevant with DMA.

25.2.46 HAL_HASH_ErrorCallback

Function Name	void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)
Function Description	Data transfer Error callback.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

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_SH224_Start_DMA())
 - Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
 - Configure and enable one DMA stream one for managing data transfer from memory to peripheral (input stream). Managing data transfer from peripheral to memory can be performed only using CPU
 - Associate the initialized DMA handle to the HASH DMA handle using __HAL_LINKDMA()
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Stream: HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HASH HAL using HAL_HASH_Init(). This function configures mainly:
 - a. The data type: 1-bit, 8-bit, 16-bit and 32-bit.
 - b. For HMAC, the encryption key.
 - c. For HMAC, the key size used for encryption.
3. Three processing functions are available:
 - a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished e.g.
HAL_HASHEx_SHA224_Start()
 - b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt e.g. HAL_HASHEx_SHA224_Start_IT()
 - c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA e.g.
HAL_HASHEx_SHA224_Start_DMA()
4. When the processing function is called at first time after HAL_HASH_Init() the HASH peripheral is initialized and processes the buffer in input. After that, the digest computation is started. When processing multi-buffer use the accumulate function to write the data in the peripheral without starting the digest computation. In last buffer use the start function to input the last buffer and start the digest computation.
 - a. e.g. HAL_HASHEx_SHA224_Accumulate() : write 1st data buffer in the peripheral without starting the digest computation
 - b. write (n-1)th data buffer in the peripheral without starting the digest computation
 - c. HAL_HASHEx_SHA224_Start() : write (n)th data buffer in the peripheral and start the digest computation
5. In HMAC mode, there is no Accumulate API. Only Start API is available.
6. In case of using DMA, call the DMA start processing e.g.
HAL_HASHEx_SHA224_Start_DMA(). After that, call the finish function in order to get the digest value e.g. HAL_HASHEx_SHA224_Finish()

-
7. Call HAL_HASH_DeInit() to deinitialize the HASH peripheral.

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	<code>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

26.1.8 HAL_HASHEx_SHA256_Start

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

26.1.9 HAL_HASHEx_SHA224_Accumulate

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA224_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
---------------	--

Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

26.1.10 HAL_HASHEx_SHA256_Accumulate

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Accumulate(HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

26.1.11 HAL_HMACEx_SHA224_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA224_Start(HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> 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. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

26.1.12 HAL_HMACEx_SHA256_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA256_Start(HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> 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.
- **Timeout:** Timeout value

Return values

- HAL status

26.1.13 HAL_HASHEx_SHA224_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)</code>
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

26.1.14 HAL_HASHEx_SHA256_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)</code>
Function Description	Initializes the HASH peripheral in SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

26.1.15 HAL_HASHEx_IRQHandler

Function Name	<code>void HAL_HASHEx_IRQHandler (HASH_HandleTypeDef * hhash)</code>
Function Description	This function handles HASH interrupt request.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none"> • None

26.1.16 HAL_HASHEx_SHA224_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in SHA224 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

26.1.17 HAL_HASHEx_SHA224_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Returns the computed digest in SHA224.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module • pOutBuffer: Pointer to the computed digest. Its size must be 28 bytes. • Timeout: Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

26.1.18 HAL_HASHEx_SHA256_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in SHA256 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

26.1.19 HAL_HASHEx_SHA256_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Returns the computed digest in SHA256.
Parameters	<ul style="list-style-type: none"> • hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module

- **pOutBuffer:** Pointer to the computed digest. Its size must be 32 bytes.
- **Timeout:** Timeout value
- HAL status

Return values

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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

26.1.26 HAL_HMACEx_SHA224_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> 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. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

26.1.27 HAL_HMACEx_SHA256_Start

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA256 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> 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. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

26.1.28 HAL_HASHEx_SHA224_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)</code>
Function Description	Initializes the HASH peripheral in SHA224 mode then processes pInBuffer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

26.1.29 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

26.1.30 HAL_HASHEx_SHA224_Start_DMA

Function Name	HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA (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"> 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	<ul style="list-style-type: none"> HAL status

26.1.31 HAL_HASHEx_SHA224_Finish

Function Name	HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)
Function Description	Returns the computed digest in SHA224.
Parameters	<ul style="list-style-type: none"> hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module pOutBuffer: Pointer to the computed digest. Its size must be 28 bytes. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

26.1.32 HAL_HASHEx_SHA256_Start_DMA

Function Name	HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA (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"> 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	<ul style="list-style-type: none"> HAL status

26.1.33 HAL_HASHEx_SHA256_Finish

Function Name	<code>HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)</code>
Function Description	Returns the computed digest in SHA256.
Parameters	<ul style="list-style-type: none"> hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module pOutBuffer: Pointer to the computed digest. Its size must be 32 bytes. Timeout: Timeout value
Return values	<ul style="list-style-type: none"> HAL status

26.1.34 HAL_HMACEx_SHA224_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA224 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

26.1.35 HAL_HMACEx_SHA256_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)</code>
Function Description	Initializes the HASH peripheral in HMAC SHA256 mode then enables DMA to control data transfer.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

26.1.36 HAL_HASHEx_IRQHandler

Function Name	<code>void HAL_HASHEx_IRQHandler (HASH_HandleTypeDef * hhash)</code>
Function Description	This function handles HASH interrupt request.
Parameters	<ul style="list-style-type: none">• hhash: pointer to a HASH_HandleTypeDef structure that contains the configuration information for HASH module
Return values	<ul style="list-style-type: none">• None

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_HCTypedef hc*
- *HAL_LockTypeDef Lock*
- *__IO HCD_StateTypeDef State*
- *void * pData*

Field Documentation

- ***HCD_TypeDef* HCD_HandleTypeDef::Instance***
Register base address
- ***HCD_InitTypeDef HCD_HandleTypeDef::Init***
HCD required parameters
- ***HCD_HCTypedef HCD_HandleTypeDef::hc[15]***
Host channels parameters
- ***HAL_LockTypeDef HCD_HandleTypeDef::Lock***
HCD peripheral status
- ***__IO HCD_StateTypeDef HCD_HandleTypeDef::State***
HCD communication state
- ***void* HCD_HandleTypeDef::pData***
Pointer Stack Handler

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_DelInit()`
- `HAL_HCD_MspInit()`
- `HAL_HCD_MspDelInit()`

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 `HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)`

Function Description Initialize the host driver.

Parameters • `hhcd`: HCD handle

Return values • HAL status

27.2.7 HAL_HCD_HC_Init

Function Name	<code>HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDefDef * hhcd, uint8_t ch_num, uint8_t eenum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)</code>
Function Description	Initialize a host channel.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle • ch_num: Channel number. This parameter can be a value from 1 to 15 • enum: Endpoint number. This parameter can be a value from 1 to 15 • dev_address: : Current device address This parameter can be a value from 0 to 255 • speed: Current device speed. This parameter can be one of these values: HCD_SPEED_HIGH: High speed mode, HCD_SPEED_FULL: Full speed mode, HCD_SPEED_LOW: Low speed mode • ep_type: Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type, EP_TYPE_ISOC: Isochronous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type • mps: Max Packet Size. This parameter can be a value from 0 to32K
Return values	<ul style="list-style-type: none"> • HAL status

27.2.8 HAL_HCD_HC_Halt

Function Name	<code>HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDefDef * hhcd, uint8_t ch_num)</code>
Function Description	Halt a host channel.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle • ch_num: Channel number. This parameter can be a value from 1 to 15
Return values	<ul style="list-style-type: none"> • HAL status

27.2.9 HAL_HCD_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_HCD_DelInit (HCD_HandleTypeDefDef * hhcd)</code>
Function Description	Deinitialize the host driver.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle
Return values	<ul style="list-style-type: none"> • HAL status

27.2.10 HAL_HCD_MspInit

Function Name	<code>void HAL_HCD_MspInit (HCD_HandleTypeDefDef * hhcd)</code>
Function Description	Initializes the HCD MSP.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle

Return values	<ul style="list-style-type: none"> None
---------------	--

27.2.11 HAL_HCD_MspDeInit

Function Name	void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)
Function Description	DeInitializes HCD MSP.
Parameters	<ul style="list-style-type: none"> hhcd: HCD handle
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hhcd: HCD handle
Return values	<ul style="list-style-type: none"> None

27.2.14 HAL_HCD_SOF_Callback

Function Name	void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)
Function Description	SOF callback.
Parameters	<ul style="list-style-type: none"> hhcd: HCD handle
Return values	<ul style="list-style-type: none"> None

27.2.15 HAL_HCD_Connect_Callback

Function Name	<code>void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)</code>
Function Description	Connexion Event callback.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle
Return values	<ul style="list-style-type: none"> • None

27.2.16 HAL_HCD_Disconnect_Callback

Function Name	<code>void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)</code>
Function Description	Disconnection Event callback.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle
Return values	<ul style="list-style-type: none"> • None

27.2.17 HAL_HCD_HC_NotifyURBChange_Callback

Function Name	<code>void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)</code>
Function Description	Notify URB state change callback.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle • chnum: Channel number. This parameter can be a value from 1 to 15 • urb_state: This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/
Return values	<ul style="list-style-type: none"> • None

27.2.18 HAL_HCD_Start

Function Name	<code>HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * hhcd)</code>
Function Description	Start the host driver.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle
Return values	<ul style="list-style-type: none"> • HAL status

27.2.19 HAL_HCD_Stop

Function Name	<code>HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * hhcd)</code>
Function Description	Stop the host driver.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle
Return values	<ul style="list-style-type: none"> • HAL status

27.2.20 HAL_HCD_ResetPort

Function Name	HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * hhcd)
Function Description	Reset the host port.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle
Return values	<ul style="list-style-type: none"> • HAL status

27.2.21 HAL_HCD_GetState

Function Name	HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef * hhcd)
Function Description	Return the HCD state.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle
Return values	<ul style="list-style-type: none"> • HAL state

27.2.22 HAL_HCD_HC_GetURBState

Function Name	HCD_URBStateTypeDef HAL_HCD_HC_GetURBState (HCD_HandleTypeDef * hhcd, uint8_t chnum)
Function Description	Return URB state for a channel.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle • chnum: Channel number. This parameter can be a value from 1 to 15
Return values	<ul style="list-style-type: none"> • URB state. This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/

27.2.23 HAL_HCD_HC_GetXferCount

Function Name	uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef * hhcd, uint8_t chnum)
Function Description	Return the last host transfer size.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle • chnum: Channel number. This parameter can be a value from 1 to 15
Return values	<ul style="list-style-type: none"> • last transfer size in byte

27.2.24 HAL_HCD_HC_GetState

Function Name	HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef * hhcd, uint8_t chnum)
Function Description	Return the Host Channel state.
Parameters	<ul style="list-style-type: none"> • hhcd: HCD handle • chnum: Channel number. This parameter can be a value from 1 to 15

Return values	<ul style="list-style-type: none"> 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	<code>uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)</code>
Function Description	Return the current Host frame number.
Parameters	<ul style="list-style-type: none"> hhcd: HCD handle
Return values	<ul style="list-style-type: none"> Current Host frame number

27.2.26 HAL_HCD_GetCurrentSpeed

Function Name	<code>uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)</code>
Function Description	Return the Host enumeration speed.
Parameters	<ul style="list-style-type: none"> hhcd: HCD handle
Return values	<ul style="list-style-type: none"> 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_DelInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- [**HAL_I2C_Init\(\)**](#)
- [**HAL_I2C_DelInit\(\)**](#)
- [**HAL_I2C_MspInit\(\)**](#)
- [**HAL_I2C_MspDelInit\(\)**](#)

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	HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)
Function Description	Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • HAL status

28.2.6 HAL_I2C_DeInit

Function Name	HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)
Function Description	Deinitializes the I2C peripheral.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • HAL status

28.2.7 HAL_I2C_MspInit

Function Name	void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)
Function Description	I2C MSP Init.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.8 HAL_I2C_MspDeInit

Function Name	void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)
Function Description	I2C MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.9 HAL_I2C_Master_Transmit

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in master mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

28.2.10 HAL_I2C_Master_Receive

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives in master mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

28.2.11 HAL_I2C_Slave_Transmit

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in slave mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

28.2.12 HAL_I2C_Slave_Receive

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive in slave mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration
- HAL status

Return values

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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

28.2.16 HAL_I2C_Slave_Receive_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT
---------------	---

(I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function Description	Receive in slave mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

28.2.17 HAL_I2C_Master_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)
Function Description	Transmit in master mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

28.2.18 HAL_I2C_Master_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)
Function Description	Receive in master mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

28.2.19 HAL_I2C_Slave_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)
Function Description	Transmit in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

28.2.23 HAL_I2C_Mem_Write_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</code>
Function Description	Write an amount of data in no-blocking mode with Interrupt to a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

28.2.24 HAL_I2C_Mem_Read_IT

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</code>
Function Description	Read an amount of data in no-blocking mode with Interrupt from a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

28.2.25 HAL_I2C_Mem_Write_DMA

Function Name	<code>HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)</code>
Function Description	Write an amount of data in no-blocking mode with DMA to a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C. • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be sent

Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values	<ul style="list-style-type: none"> • None
---------------	--

28.2.30 HAL_I2C_MasterTxCpltCallback

Function Name	void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Master Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.31 HAL_I2C_MasterRxCpltCallback

Function Name	void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Master Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.32 HAL_I2C_SlaveTxCpltCallback

Function Name	void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.33 HAL_I2C_SlaveRxCpltCallback

Function Name	void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Slave Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.34 HAL_I2C_MemTxCpltCallback

Function Name	void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Memory Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.35 HAL_I2C_MemRxCpltCallback

Function Name	<code>void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)</code>
Function Description	Memory Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.36 HAL_I2C_ErrorCallback

Function Name	<code>void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)</code>
Function Description	I2C error callbacks.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • None

28.2.37 HAL_I2C_GetState

Function Name	<code>HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)</code>
Function Description	Returns the I2C state.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • HAL state

28.2.38 HAL_I2C_GetError

Function Name	<code>uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)</code>
Function Description	Return the I2C error code.
Parameters	<ul style="list-style-type: none"> • hi2c: : Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> • I2C Error Code

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

<code>_HAL_I2C_RESET_HANDLE_STATE</code>	Description: <ul style="list-style-type: none"> Reset I2C handle state. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: specifies the I2C Handle. Return value: <ul style="list-style-type: none"> None
<code>_HAL_I2C_ENABLE_IT</code>	Description: <ul style="list-style-type: none"> Enable the specified I2C interrupts. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: specifies the I2C Handle. <code>_INTERRUPT_</code>: specifies the interrupt source to enable. This parameter can be one of the following values: <ul style="list-style-type: none"> <code>I2C_IT_ERRI</code>: Errors interrupt enable <code>I2C_IT_TCI</code>: Transfer complete interrupt enable <code>I2C_IT_STOPI</code>: STOP detection interrupt enable <code>I2C_IT_NACKI</code>: NACK received interrupt enable <code>I2C_IT_ADDRI</code>: Address match interrupt enable <code>I2C_IT_RXI</code>: RX interrupt enable <code>I2C_IT_TXI</code>: TX interrupt enable Return value: <ul style="list-style-type: none"> None
<code>_HAL_I2C_DISABLE_IT</code>	Description: <ul style="list-style-type: none"> Disable the specified I2C interrupts. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: specifies the I2C Handle. <code>_INTERRUPT_</code>: specifies the interrupt source to disable. This parameter can be one of the following values: <ul style="list-style-type: none"> <code>I2C_IT_ERRI</code>: Errors interrupt enable <code>I2C_IT_TCI</code>: Transfer complete interrupt enable <code>I2C_IT_STOPI</code>: STOP detection interrupt enable <code>I2C_IT_NACKI</code>: NACK received interrupt enable <code>I2C_IT_ADDRI</code>: Address match interrupt enable <code>I2C_IT_RXI</code>: RX interrupt enable <code>I2C_IT_TXI</code>: TX interrupt enable

- 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:

- None

`_HAL_I2C_ENABLE`

Description:

- Enable the specified I2C peripheral.

Parameters:

- `_HANDLE_`: specifies the I2C Handle.

Return value:

- None

`_HAL_I2C_DISABLE`

Description:

- Disable the specified I2C peripheral.

Parameters:

- `_HANDLE_`: specifies the I2C Handle.

Return value:

- None

I2C Flag definition

`I2C_FLAG_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 I2CEEx 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_I2CEEx_ConfigAnalogFilter()`
2. Configure I2C Digital noise filter using the function `HAL_I2CEEx_ConfigDigitalFilter()`

29.1.3 Extended features functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- [`HAL_I2CEEx_ConfigAnalogFilter\(\)`](#)
- [`HAL_I2CEEx_ConfigDigitalFilter\(\)`](#)

29.1.4 `HAL_I2CEEx_ConfigAnalogFilter`

Function Name	<code>HAL_StatusTypeDef HAL_I2CEEx_ConfigAnalogFilter(I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)</code>
Function Description	Configures I2C Analog noise filter.
Parameters	<ul style="list-style-type: none">• hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.• AnalogFilter: : new state of the Analog filter.
Return values	<ul style="list-style-type: none">• HAL status

29.1.5 `HAL_I2CEEx_ConfigDigitalFilter`

Function Name	<code>HAL_StatusTypeDef HAL_I2CEEx_ConfigDigitalFilter(I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)</code>
Function Description	Configures I2C Digital noise filter.
Parameters	<ul style="list-style-type: none">• hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.• DigitalFilter: : Coefficient of digital noise filter between 0x00 and 0x0F.

Return values

- HAL status

29.2 I2CEEx Firmware driver defines

29.2.1 I2CEEx

I2CEEx 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 SYSCLK or External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32f3xx_hal_conf.h file.

4. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()
- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop()

I2S HAL driver macros list

Below the list of most used macros in I2S HAL driver.

- `_HAL_I2S_ENABLE`: Enable the specified SPI peripheral (in I2S mode)
- `_HAL_I2S_DISABLE`: Disable the specified SPI peripheral (in I2S mode)
- `_HAL_I2S_ENABLE_IT` : Enable the specified I2S interrupts
- `_HAL_I2S_DISABLE_IT` : Disable the specified I2S interrupts
- `_HAL_I2S_GET_FLAG`: Check whether the specified I2S flag is set or not



You can refer to the I2S HAL driver header file for more useful macros

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	HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)
Function Description	Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status

30.2.6 HAL_I2S_DeInit

Function Name	HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)
Function Description	Deinitializes the I2S peripheral.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

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

30.2.7 HAL_I2S_MspInit

Function Name	void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)
Function Description	I2S MSP Init.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None

30.2.8 HAL_I2S_MspDeInit

Function Name	void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)
Function Description	I2S MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to data buffer. • Size: number of data sample to be sent:

- **Timeout:** Timeout duration
- Return values**
- HAL status
- Notes**
- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the **Size** parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the **Size** parameter means the number of 16-bit data length.
 - The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
 - In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction.

30.2.11 HAL_I2S_Transmit_IT

- | | |
|-----------------------------|---|
| Function Name | HAL_StatusTypeDef HAL_I2S_Transmit_IT
(I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size) |
| Function Description | Transmit an amount of data in non-blocking mode with Interrupt. |
| Parameters | <ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to data buffer. • Size: number of data sample to be sent: |
| Return values | <ul style="list-style-type: none"> • HAL status |
| Notes | <ul style="list-style-type: none"> • 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). |

30.2.12 HAL_I2S_Receive_IT

- | | |
|-----------------------------|--|
| Function Name | HAL_StatusTypeDef HAL_I2S_Receive_IT
(I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size) |
| Function Description | Receive an amount of data in non-blocking mode with Interrupt. |
| Parameters | <ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Receive data buffer. • Size: number of data sample to be sent: |
| Return values | <ul style="list-style-type: none"> • HAL status |
| Notes | <ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data |

frame is selected the Size parameter means the number of 16-bit data length.

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- It is recommended to use DMA for the I2S receiver to avoid de-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.

30.2.13 HAL_I2S_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_I2S_Transmit_DMA(I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Transmit data buffer. • Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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).

30.2.14 HAL_I2S_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_I2S_Receive_DMA(I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Receive data buffer. • Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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).

30.2.15 HAL_I2S_DMAPause



Function Name	HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)
Function Description	Pauses the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status

30.2.16 HAL_I2S_DMAResume

Function Name	HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status

30.2.17 HAL_I2S_DMAStop

Function Name	HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)
Function Description	Stops the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status

30.2.18 HAL_I2S_IRQHandler

Function Name	void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)
Function Description	This function handles I2S interrupt request.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status

30.2.19 HAL_I2S_TxHalfCpltCallback

Function Name	void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Tx Transfer Half completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None

30.2.20 HAL_I2S_TxCpltCallback

Function Name	void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Tx Transfer completed callbacks.

Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None

30.2.21 HAL_I2S_RxHalfCpltCallback

Function Name	void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Rx Transfer half completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None

30.2.22 HAL_I2S_RxCpltCallback

Function Name	void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None

30.2.23 HAL_I2S_ErrorCallback

Function Name	void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)
Function Description	I2S error callbacks.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • None

30.2.24 HAL_I2S_GetState

Function Name	HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)
Function Description	Return the I2S state.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL state

30.2.25 HAL_I2S_GetError

Function Name	uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)
Function Description	Return the I2S error code.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • I2S Error Code

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

- | <u>__HAL_I2S_RESET_HANDLE_STATE</u> | Description: |
|-------------------------------------|--|
| | <ul style="list-style-type: none">• Reset I2S handle state. |
| | Parameters: |
| | <ul style="list-style-type: none">• <u>__HANDLE__</u>: 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_DelInit\(\)**](#)
- [**HAL_IRDA_MspInit\(\)**](#)
- [**HAL_IRDA_MspDelInit\(\)**](#)

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_DMAPause\(\)***](#)
- [***HAL_IRDA_DMAResume\(\)***](#)
- [***HAL_IRDA_DMASuspend\(\)***](#)
- [***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	HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)
Function Description	Initializes the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • HAL status

31.2.6 HAL_IRDA_DeInit

Function Name	HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)
Function Description	DeInitializes the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA

module.

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

31.2.7 HAL_IRDA_MspInit

Function Name	void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)
Function Description	IRDA MSP Init.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

31.2.8 HAL_IRDA_MspDeInit

Function Name	void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)
Function Description	IRDA MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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"> • HAL status
Notes	<ul style="list-style-type: none"> • When the IRDA parity is enabled (PCE = 1) the data received contain the parity bit.

31.2.15 HAL_IRDA_DMAPause

Function Name	HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • HAL status

31.2.16 HAL_IRDA_DMAResume

Function Name	HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • HAL status

31.2.17 HAL_IRDA_DMAStop

Function Name	HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • HAL status

31.2.18 HAL_IRDA_IRQHandler

Function Name	void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)
Function Description	This function handles IRDA interrupt request.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

31.2.19 HAL_IRDA_TxHalfCpltCallback

Function Name	void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)
---------------	--

Function Description	Tx Transfer complete callbacks.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

31.2.20 HAL_IRDA_TxCpltCallback

Function Name	void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none"> • None

31.2.21 HAL_IRDA_RxHalfCpltCallback

Function Name	void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Rx Transfer complete callbacks.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

31.2.22 HAL_IRDA_RxCpltCallback

Function Name	void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Rx Half Transfer complete callbacks.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

31.2.23 HAL_IRDA_ErrorCallback

Function Name	void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)
Function Description	IRDA error callbacks.
Parameters	<ul style="list-style-type: none"> • hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • None

31.2.24 HAL_IRDA_GetState

Function Name	HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)
Function Description	Returns the IRDA state.
Parameters	<ul style="list-style-type: none"> hirda: pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> HAL state

31.2.25 HAL_IRDA_GetError

Function Name	uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)
Function Description	Return the IRDA error code.
Parameters	<ul style="list-style-type: none"> hirda: : pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA.
Return values	<ul style="list-style-type: none"> IRDA Error Code

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

<code>__HAL_IRDA_RESET_HANDLE_STA TE</code>	Description: <ul style="list-style-type: none"> Reset IRDA handle state. Parameters: <ul style="list-style-type: none"> <code>__HANDLE__</code>: specifies the IRDA Handle. The Handle Instance which can be USART1 or USART2. Return value:
---	---

- None

_HAL_IRDA_GET_FLAG

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_RXACK: 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

_HAL_IRDA_DISABLE_IT

- 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

_HAL_IRDA_GET_IT

- 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

- IRDA_IT_ORE: OverRun Error interrupt
- IRDA_IT_NE: Noise Error interrupt
- IRDA_IT_FE: Framing Error interrupt
- IRDA_IT_PE: Parity Error interrupt

Return value:

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

__HAL_IRDA_GET_IT_SOURCE

Description:

- 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).

__HAL_IRDA_CLEAR_IT

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

- Clear Flag
 - IRDA_CLEAR_TCF: Transmission Complete Clear Flag

Return value:

- None

_HAL_IRDA_SEND_REQ**Description:**

- Set a specific IRDA request flag.

Parameters:

- HANDLE: specifies the IRDA Handle. The Handle Instance 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

_HAL_IRDA_ENABLE**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

_HAL_IRDA_DISABLE**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

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_HandleTypeDefDef

Data Fields

- *IWDG_TypeDef * Instance*
- *IWDG_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_IWDG_StateTypeDef State*

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDefDef::Instance***
Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDefDef::Init***
IWDG required parameters
- ***HAL_LockTypeDef IWDG_HandleTypeDefDef::Lock***
IWDG Locking object
- ***__IO HAL_IWDG_StateTypeDef IWDG_HandleTypeDefDef::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_MspInit\(\)*](#)

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	<code>HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)</code>
Function Description	Initializes the IWDG according to the specified parameters in the IWDG_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

33.2.5 HAL_IWDG_MspInit

Function Name	<code>void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)</code>
Function Description	Initializes the IWDG MSP.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • None

33.2.6 HAL_IWDG_Start

Function Name	HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)
Function Description	Starts the IWDG.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

33.2.7 HAL_IWDG_Refresh

Function Name	HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)
Function Description	Refreshes the IWDG.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

33.2.8 HAL_IWDG_GetState

Function Name	HAL_IWDG_StateTypeDef HAL_IWDG_GetState (IWDG_HandleTypeDef * hiwdg)
Function Description	Returns the IWDG state.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL state

33.3 IWDG Firmware driver defines

33.3.1 IWDG

IWDG Exported Macros

<u>__HAL_IWDG_RESET_HANDLE_STATE</u>	Description:
	<ul style="list-style-type: none"> • Reset IWDG handle state.
	Parameters:
	<ul style="list-style-type: none"> • <u>__HANDLE__</u>: IWDG handle.
	Return value:
	<ul style="list-style-type: none"> • None
<u>__HAL_IWDG_START</u>	Description:
	<ul style="list-style-type: none"> • Enables the IWDG peripheral.
	Parameters:

- `__HANDLE__`: IWDG handle

Return value:

- None

`__HAL_IWDG_RELOAD_COUNTER`**Description:**

- Reloads IWDG counter with value defined in the reload register (write access to IWDG_PR and IWDG_RLR registers disabled).

Parameters:

- `__HANDLE__`: IWDG handle

Return value:

- None

`__HAL_IWDG_GET_FLAG`**Description:**

- Gets the selected IWDG's flag status.

Parameters:

- `__HANDLE__`: IWDG handle
- `__FLAG__`: specifies the flag to check.
This parameter can be one of the following values:
 - IWDG_FLAG_PVU: Watchdog counter reload value update flag
 - IWDG_FLAG_RVU: Watchdog counter prescaler value flag
 - IWDG_FLAG_WVU: Watchdog counter window value flag

Return value:

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

IWDG Prescaler

<code>IWDG_PRESCALER_4</code>	IWDG prescaler set to 4
<code>IWDG_PRESCALER_8</code>	IWDG prescaler set to 8
<code>IWDG_PRESCALER_16</code>	IWDG prescaler set to 16
<code>IWDG_PRESCALER_32</code>	IWDG prescaler set to 32
<code>IWDG_PRESCALER_64</code>	IWDG prescaler set to 64
<code>IWDG_PRESCALER_128</code>	IWDG prescaler set to 128
<code>IWDG_PRESCALER_256</code>	IWDG prescaler set to 256

IWDG Private Defines`HAL_IWDG_DEFAULT_TIMEOUT`

<code>IWDG_KEY_RELOAD</code>	IWDG Reload Counter Enable
<code>IWDG_KEY_ENABLE</code>	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	Description: <ul style="list-style-type: none"> Enables write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers. Parameters: <ul style="list-style-type: none"> <code>__HANDLE__</code>: IWDG handle Return value: <ul style="list-style-type: none"> None
IWDG_DISABLE_WRITE_ACCESS	Description: <ul style="list-style-type: none"> Disables write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers. Parameters: <ul style="list-style-type: none"> <code>__HANDLE__</code>: IWDG handle Return value: <ul style="list-style-type: none"> None
IS_IWDG_PRESCALER	Description: <ul style="list-style-type: none"> Check IWDG prescaler value. Parameters: <ul style="list-style-type: none"> <code>__PRESCALER__</code>: IWDG prescaler value Return value: <ul style="list-style-type: none"> None
IS_IWDG_RELOAD	Description: <ul style="list-style-type: none"> Check IWDG reload value. Parameters: <ul style="list-style-type: none"> <code>__RELOAD__</code>: IWDG reload value Return value: <ul style="list-style-type: none"> None
IS_IWDG_WINDOW	Description: <ul style="list-style-type: none"> Check IWDG window value. Parameters: <ul style="list-style-type: none"> <code>__WINDOW__</code>: IWDG window value

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 possible.
 - 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 possible.
 - 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_Delnit() 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_Delnit\(\)*](#)
- [*HAL_LPTIM_MspInit\(\)*](#)
- [*HAL_LPTIM_MspDelnit\(\)*](#)

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 **HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hltim)**

Function Description Initializes the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and creates the associated handle.

Parameters	<ul style="list-style-type: none"> hlptim: LPTIM handle
Return values	<ul style="list-style-type: none"> HAL status

34.2.8 HAL_LPTIM_DelInit

Function Name	HAL_StatusTypeDef HAL_LPTIM_DelInit (LPTIM_HandleTypeDef * hlptim)
Function Description	DeInitializes the LPTIM peripheral.
Parameters	<ul style="list-style-type: none"> hlptim: LPTIM handle
Return values	<ul style="list-style-type: none"> HAL status

34.2.9 HAL_LPTIM_MspInit

Function Name	void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)
Function Description	Initializes the LPTIM MSP.
Parameters	<ul style="list-style-type: none"> hlptim: LPTIM handle
Return values	<ul style="list-style-type: none"> None

34.2.10 HAL_LPTIM_MspDeInit

Function Name	void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)
Function Description	DeInitializes LPTIM MSP.
Parameters	<ul style="list-style-type: none"> hlptim: LPTIM handle
Return values	<ul style="list-style-type: none"> None

34.2.11 HAL_LPTIM_PWM_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Period, uint32_t Pulse)
Function Description	Starts the LPTIM PWM generation.
Parameters	<ul style="list-style-type: none"> hlptim: : LPTIM handle Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> HAL status

34.2.12 HAL_LPTIM_PWM_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM PWM generation.
Parameters	<ul style="list-style-type: none"> hlptim: : LPTIM handle
Return values	<ul style="list-style-type: none"> HAL status

34.2.13 HAL_LPTIM_PWM_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)</code>
Function Description	Starts the LPTIM PWM generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF • Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF
Return values	<ul style="list-style-type: none"> • HAL status

34.2.14 HAL_LPTIM_PWM_Stop_IT

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hltim)</code>
Function Description	Stops the LPTIM PWM generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.15 HAL_LPTIM_OnePulse_Start

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)</code>
Function Description	Starts the LPTIM One pulse generation.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.16 HAL_LPTIM_OnePulse_Stop

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hltim)</code>
Function Description	Stops the LPTIM One pulse generation.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.17 HAL_LPTIM_OnePulse_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Pulse)</code>
---------------	---

Function Description	Starts the LPTIM One pulse generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.18 HAL_LPTIM_OnePulse_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM One pulse generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hlptim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.19 HAL_LPTIM_SetOnce_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (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"> • hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse: : Specifies the compare value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.20 HAL_LPTIM_SetOnce_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim)
Function Description	Stops the LPTIM Set once mode.
Parameters	<ul style="list-style-type: none"> • hlptim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.21 HAL_LPTIM_SetOnce_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (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"> • hlptim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF. • Pulse: : Specifies the compare value. This parameter must

be a value between 0x0000 and 0xFFFF.

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

34.2.22 HAL_LPTIM_SetOnce_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the LPTIM Set once mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.23 HAL_LPTIM_Encoder_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period)
Function Description	Starts the Encoder interface.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.24 HAL_LPTIM_Encoder_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Encoder interface.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.25 HAL_LPTIM_Encoder_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period)
Function Description	Starts the Encoder interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.26 HAL_LPTIM_Encoder_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Encoder interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle

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

34.2.27 HAL_LPTIM_TimeOut_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Timeout)
Function Description	Starts the Timeout function.
Parameters	<ul style="list-style-type: none"> • hltim: : 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	<ul style="list-style-type: none"> • HAL status

34.2.28 HAL_LPTIM_TimeOut_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Timeout function.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.29 HAL_LPTIM_TimeOut_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period, uint32_t Timeout)
Function Description	Starts the Timeout function in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : 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	<ul style="list-style-type: none"> • HAL status

34.2.30 HAL_LPTIM_TimeOut_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Timeout function in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.31 HAL_LPTIM_Counter_Start

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hltim, uint32_t Period)
Function Description	Starts the Counter mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.32 HAL_LPTIM_Counter_Stop

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Counter mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.33 HAL_LPTIM_Counter_Start_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hltim, uint32_t Period)
Function Description	Starts the Counter mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle • Period: : Specifies the Autoreload value. This parameter must be a value between 0x0000 and 0xFFFF.
Return values	<ul style="list-style-type: none"> • HAL status

34.2.34 HAL_LPTIM_Counter_Stop_IT

Function Name	HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hltim)
Function Description	Stops the Counter mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL status

34.2.35 HAL_LPTIM_ReadCounter

Function Name	uint32_t HAL_LPTIM_ReadCounter (LPTIM_HandleTypeDef * hltim)
Function Description	This function returns the current counter value.
Parameters	<ul style="list-style-type: none"> • hltim: LPTIM handle
Return values	<ul style="list-style-type: none"> • Counter value.

34.2.36 HAL_LPTIM_ReadAutoReload

Function Name	uint32_t HAL_LPTIM_ReadAutoReload (LPTIM_HandleTypeDef * hltim)
---------------	--

Function Description	This function return the current Autoreload (Period) value.
Parameters	<ul style="list-style-type: none"> • hlptim: LPTIM handle
Return values	<ul style="list-style-type: none"> • Autoreload value.

34.2.37 HAL_LPTIM_ReadCompare

Function Name	uint32_t HAL_LPTIM_ReadCompare (LPTIM_HandleTypeDef * hlptim)
Function Description	This function return the current Compare (Pulse) value.
Parameters	<ul style="list-style-type: none"> • hlptim: LPTIM handle
Return values	<ul style="list-style-type: none"> • Compare value.

34.2.38 HAL_LPTIM_IRQHandler

Function Name	void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)
Function Description	This function handles LPTIM interrupt request.
Parameters	<ul style="list-style-type: none"> • hlptim: LPTIM handle
Return values	<ul style="list-style-type: none"> • None

34.2.39 HAL_LPTIM_CompareMatchCallback

Function Name	void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Compare match callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hlptim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • None

34.2.40 HAL_LPTIM_AutoReloadMatchCallback

Function Name	void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Autoreload match callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hlptim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • None

34.2.41 HAL_LPTIM_TriggerCallback

Function Name	void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)
Function Description	Trigger detected callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hlptim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • None

34.2.42 HAL_LPTIM_CompareWriteCallback

Function Name	void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hltim)
Function Description	Compare write callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • None

34.2.43 HAL_LPTIM_AutoReloadWriteCallback

Function Name	void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hltim)
Function Description	Autoreload write callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • None

34.2.44 HAL_LPTIM_DirectionUpCallback

Function Name	void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hltim)
Function Description	Direction counter changed from Down to Up callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • None

34.2.45 HAL_LPTIM_DirectionDownCallback

Function Name	void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hltim)
Function Description	Direction counter changed from Up to Down callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hltim: : LPTIM handle
Return values	<ul style="list-style-type: none"> • None

34.2.46 HAL_LPTIM_GetState

Function Name	HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (LPTIM_HandleTypeDef * hltim)
Function Description	Returns the LPTIM state.
Parameters	<ul style="list-style-type: none"> • hltim: LPTIM handle
Return values	<ul style="list-style-type: none"> • HAL state

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_APBCLOCK_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).

_HAL_LPTIM_CLEAR_FLAG**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.

_HAL_LPTIM_ENABLE_IT**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

Interrupt.

Return value:

- None.

`__HAL_LPTIM_DISABLE_IT`

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.

`__HAL_LPTIM_GET_IT_SOURCE`

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:

- 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 PCPPolarity*
- *uint32_t HorizontalSync*
- *uint32_t VerticalSync*
- *uint32_t AccumulatedHBP*
- *uint32_t AccumulatedVBP*
- *uint32_t AccumulatedActiveW*
- *uint32_t AccumulatedActiveH*
- *uint32_t TotalWidth*
- *uint32_t TotalHeigh*
- *LTDC_ColorTypeDef Backcolor*

Field Documentation

- ***uint32_t LTDC_InitTypeDef::HSPolarity***
configures the horizontal synchronization polarity. This parameter can be one value of ***LTDC_HS_POLARITY***
- ***uint32_t LTDC_InitTypeDef::VSPolarity***
configures the vertical synchronization polarity. This parameter can be one value of ***LTDC_VS_POLARITY***
- ***uint32_t LTDC_InitTypeDef::DEPolarity***
configures the data enable polarity. This parameter can be one of value of ***LTDC_DE_POLARITY***
- ***uint32_t LTDC_InitTypeDef::PCPolarity***
configures the pixel clock polarity. This parameter can be one of value of ***LTDC_PC_POLARITY***
- ***uint32_t LTDC_InitTypeDef::HorizontalSync***
configures the number of Horizontal synchronization width. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.
- ***uint32_t LTDC_InitTypeDef::VerticalSync***
configures the number of Vertical synchronization 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 = 0xFFFF.
- ***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 = 0xFFFF.
- ***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 = 0xFFFF.
- ***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 FBStartAdress*
- *uint32_t ImageWidth*
- *uint32_t ImageHeight*
- *LTDC_ColorTypeDef Backcolor*

Field Documentation

- *uint32_t LTDC_LayerCfgTypeDef::WindowX0*
Configures the Window Horizontal Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.
- *uint32_t LTDC_LayerCfgTypeDef::WindowX1*
Configures the Window Horizontal Stop Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.
- *uint32_t LTDC_LayerCfgTypeDef::WindowY0*
Configures the Window vertical Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.
- *uint32_t LTDC_LayerCfgTypeDef::WindowY1*
Configures the Window vertical Stop Position. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- *uint32_t LTDC_LayerCfgTypeDef::PixelFormat*
Specifies the pixel format. This parameter can be one of value of [**LTDC_PixelFormat**](#)
- *uint32_t LTDC_LayerCfgTypeDef::Alpha*
Specifies the constant alpha used for blending. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- *uint32_t LTDC_LayerCfgTypeDef::Alpha0*
Configures the default alpha value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- *uint32_t LTDC_LayerCfgTypeDef::BlendingFactor1*
Select the blending factor 1. This parameter can be one of value of [**LTDC_BlendingFactor1**](#)
- *uint32_t LTDC_LayerCfgTypeDef::BlendingFactor2*
Select the blending factor 2. This parameter can be one of value of [**LTDC_BlendingFactor2**](#)
- *uint32_t LTDC_LayerCfgTypeDef::FBStartAdress*
Configures the color frame buffer address
- *uint32_t LTDC_LayerCfgTypeDef::ImageWidth*
Configures the color frame buffer line length. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x1FFF.
- *uint32_t LTDC_LayerCfgTypeDef::ImageHeight*
Specifies the number of line in frame buffer. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- *LTDC_ColorTypeDef LTDC_LayerCfgTypeDef::Backcolor*
Configures the layer background color.

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_DelInit()`
- `HAL_LTDC_MspInit()`
- `HAL_LTDC_MspDelInit()`
- `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	<code>HAL_StatusTypeDef HAL_LTDC_Init (LTDC_HandleTypeDef * hltc)</code>
Function Description	Initializes the LTDC according to the specified parameters in the <code>LTDC_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a <code>LTDC_HandleTypeDef</code> structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> • HAL status

35.2.7 HAL_LTDC_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_LTDC_DeInit (LTDC_HandleTypeDef * hltc)</code>
Function Description	Deinitializes the LTDC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a <code>LTDC_HandleTypeDef</code> structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> • None

35.2.8 HAL_LTDC_MspInit

Function Name	<code>void HAL_LTDC_MspInit (LTDC_HandleTypeDef * hltc)</code>
Function Description	Initializes the LTDC MSP.
Parameters	<ul style="list-style-type: none"> • hltc: : pointer to a <code>LTDC_HandleTypeDef</code> structure that contains the configuration information for the LTDC.

Return values	<ul style="list-style-type: none"> None
---------------	--

35.2.9 HAL_LTDC_MspDeInit

Function Name	void HAL_LTDC_MspDeInit (LTDC_HandleTypeDefDef * hltc)
Function Description	DeInitializes the LTDC MSP.
Parameters	<ul style="list-style-type: none"> hltc: : pointer to a LTDC_HandleTypeDefDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> None

35.2.10 HAL_LTDC_ErrorCallback

Function Name	void HAL_LTDC_ErrorCallback (LTDC_HandleTypeDefDef * hltc)
Function Description	Error LTDC callback.
Parameters	<ul style="list-style-type: none"> hltc: pointer to a LTDC_HandleTypeDefDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> None

35.2.11 HAL_LTDC_LineEvenCallback

Function Name	void HAL_LTDC_LineEvenCallback (LTDC_HandleTypeDefDef * hltc)
Function Description	Line Event callback.
Parameters	<ul style="list-style-type: none"> hltc: pointer to a LTDC_HandleTypeDefDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> None

35.2.12 HAL_LTDC_IRQHandler

Function Name	void HAL_LTDC_IRQHandler (LTDC_HandleTypeDefDef * hltc)
Function Description	Handles LTDC interrupt request.
Parameters	<ul style="list-style-type: none"> hltc: pointer to a LTDC_HandleTypeDefDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> HAL status

35.2.13 HAL_LTDC_ErrorCallback

Function Name	void HAL_LTDC_ErrorCallback (LTDC_HandleTypeDefDef * hltc)
Function Description	Error LTDC callback.
Parameters	<ul style="list-style-type: none"> hltc: pointer to a LTDC_HandleTypeDefDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> None

35.2.14 HAL_LTDC_LineEvenCallback

Function Name	void HAL_LTDC_LineEvenCallback (LTDC_HandleTypeDefDef * hltc)
---------------	--

hLcdc)

Function Description	Line Event callback.
Parameters	<ul style="list-style-type: none"> • hLcdc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> • None

35.2.15 HAL_LTDC_ConfigLayer

Function Name	HAL_StatusTypeDef HAL_LTDC_ConfigLayer (LTDC_HandleTypeDef * hLcdc, 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"> • hLcdc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • pLayerCfg: pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer. • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.16 HAL_LTDC_ConfigColorKeying

Function Name	HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying (LTDC_HandleTypeDef * hLcdc, uint32_t RGBValue, uint32_t LayerIdx)
Function Description	Configure the color keying.
Parameters	<ul style="list-style-type: none"> • hLcdc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • RGBValue: the color key value • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.17 HAL_LTDC_ConfigCLUT

Function Name	HAL_StatusTypeDef HAL_LTDC_ConfigCLUT (LTDC_HandleTypeDef * hLcdc, uint32_t * pCLUT, uint32_t CLUTSize, uint32_t LayerIdx)
Function Description	Load the color lookup table.
Parameters	<ul style="list-style-type: none"> • hLcdc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • pCLUT: pointer to the color lookup table address. • CLUTSize: the color lookup table size. • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.18 HAL_LTDC_EnableColorKeying

Function Name	HAL_StatusTypeDef HAL_LTDC_EnableColorKeying (LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)
Function Description	Enable the color keying.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.19 HAL_LTDC_DisableColorKeying

Function Name	HAL_StatusTypeDef HAL_LTDC_DisableColorKeying (LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)
Function Description	Disable the color keying.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.20 HAL_LTDC_EnableCLUT

Function Name	HAL_StatusTypeDef HAL_LTDC_EnableCLUT (LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)
Function Description	Enable the color lookup table.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.21 HAL_LTDC_DisableCLUT

Function Name	HAL_StatusTypeDef HAL_LTDC_DisableCLUT (LTDC_HandleTypeDef * hltc, uint32_t LayerIdx)
Function Description	Disable the color lookup table.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.22 HAL_LTDC_EnableDither

Function Name	HAL_StatusTypeDef HAL_LTDC_EnableDither (LTDC_HandleTypeDef * hltc)
---------------	--

Function Description	Enables Dither.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> • HAL status

35.2.23 HAL_LTDC_DisableDither

Function Name	HAL_StatusTypeDef HAL_LTDC_DisableDither (LTDC_HandleTypeDef * hltc)
Function Description	Disables Dither.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> • HAL status

35.2.24 HAL_LTDC_SetWindowSize

Function Name	HAL_StatusTypeDef HAL_LTDC_SetWindowSize (LTDC_HandleTypeDef * hltc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)
Function Description	Set the LTDC window size.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • XSize: LTDC Pixel per line • YSize: LTDC Line number • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.25 HAL_LTDC_SetWindowPosition

Function Name	HAL_StatusTypeDef HAL_LTDC_SetWindowPosition (LTDC_HandleTypeDef * hltc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)
Function Description	Set the LTDC window position.
Parameters	<ul style="list-style-type: none"> • hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. • X0: LTDC window X offset • Y0: LTDC window Y offset • LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> • HAL status

35.2.26 HAL_LTDC_SetPixelFormat

Function Name	HAL_StatusTypeDef HAL_LTDC_SetPixelFormat (LTDC_HandleTypeDef * hltc, uint32_t PixelFormat, uint32_t LayerIdx)
---------------	---

Function Description	Reconfigure the pixel format.
Parameters	<ul style="list-style-type: none"> hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. PixelFormat: new pixel format value. LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1.
Return values	<ul style="list-style-type: none"> HAL status

35.2.27 HAL_LTDC_SetAlpha

Function Name	HAL_StatusTypeDef HAL_LTDC_SetAlpha (LTDC_HandleTypeDef * hltc, uint32_t Alpha, uint32_t LayerIdx)
Function Description	Reconfigure the layer alpha value.
Parameters	<ul style="list-style-type: none"> hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. Alpha: new alpha value. LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1
Return values	<ul style="list-style-type: none"> HAL status

35.2.28 HAL_LTDC_SetAddress

Function Name	HAL_StatusTypeDef HAL_LTDC_SetAddress (LTDC_HandleTypeDef * hltc, uint32_t Address, uint32_t LayerIdx)
Function Description	Reconfigure the frame buffer Address.
Parameters	<ul style="list-style-type: none"> hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. Address: new address value. LayerIdx: LTDC Layer index. This parameter can be one of the following values: 0 or 1.
Return values	<ul style="list-style-type: none"> HAL status

35.2.29 HAL_LTDC_ProgramLineEvent

Function Name	HAL_StatusTypeDef HAL_LTDC_ProgramLineEvent (LTDC_HandleTypeDef * hltc, uint32_t Line)
Function Description	Define the position of the line interrupt .
Parameters	<ul style="list-style-type: none"> hltc: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC. Line: Line Interrupt Position.
Return values	<ul style="list-style-type: none"> HAL status

35.2.30 HAL_LTDC_GetState

Function Name	HAL_LTDC_StateTypeDef HAL_LTDC_GetState (LTDC_HandleTypeDef * hltc)
---------------	--

Function Description	Return the LTDC state.
Parameters	<ul style="list-style-type: none"> • hltcd: pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> • HAL state

35.2.31 HAL_LTDC_GetError

Function Name	uint32_t HAL_LTDC_GetError (LTDC_HandleTypeDef * hltcd)
Function Description	Return the LTDC error code.
Parameters	<ul style="list-style-type: none"> • hltcd: : pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
Return values	<ul style="list-style-type: none"> • LTDC Error Code

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

handle.

Return value:

- None

`__HAL_LTDC_ENABLE`

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**Return value:**

- None

`_HAL_LTDC_DISABLE_IT`

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_DelInit\(\)`](#)
- [`HAL_NAND_MspInit\(\)`](#)
- [`HAL_NAND_MspDelInit\(\)`](#)
- [`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	<code>HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)</code>
Function Description	Perform NAND memory Initialization sequence.
Parameters	<ul style="list-style-type: none"> • hnand: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • ComSpace_Timing: pointer to Common space timing structure • AttSpace_Timing: pointer to Attribute space timing structure
Return values	HAL status

36.2.7 HAL_NAND_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_NAND_DelInit (NAND_HandleTypeDef * hnand)</code>
Function Description	Perform NAND memory De-Initialization sequence.
Parameters	<ul style="list-style-type: none"> • hnand: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

HAL status

36.2.8 HAL_NAND_MspInit

Function Name	<code>void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)</code>
Function Description	NAND MSP Init.
Parameters	<ul style="list-style-type: none"> • hnand: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	None

36.2.9 HAL_NAND_MspDeInit

Function Name	void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hhand)
Function Description	NAND MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hhand: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	<ul style="list-style-type: none"> • None

36.2.10 HAL_NAND_IRQHandler

Function Name	void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hhand)
Function Description	This function handles NAND device interrupt request.
Parameters	<ul style="list-style-type: none"> • hhand: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	<ul style="list-style-type: none"> • HAL status

36.2.11 HAL_NAND_ITCallback

Function Name	void HAL_NAND_ITCallback (NAND_HandleTypeDef * hhand)
Function Description	NAND interrupt feature callback.
Parameters	<ul style="list-style-type: none"> • hhand: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	<ul style="list-style-type: none"> • None

36.2.12 HAL_NAND_Read_ID

Function Name	HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hhand, NAND_IDTypeDef * pNAND_ID)
Function Description	Read the NAND memory electronic signature.
Parameters	<ul style="list-style-type: none"> • hhand: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • pNAND_ID: NAND ID structure
Return values	<ul style="list-style-type: none"> • HAL status

36.2.13 HAL_NAND_Reset

Function Name	HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hhand)
Function Description	NAND memory reset.
Parameters	<ul style="list-style-type: none"> • hhand: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	<ul style="list-style-type: none"> • HAL status

36.2.14 HAL_NAND_Read_Page

Function Name	HAL_StatusTypeDef HAL_NAND_Read_Page (NAND_HandleTypeDef * hndl, 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"> hndl: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. pAddress: : pointer to NAND address structure pBuffer: : pointer to destination read buffer NumPageToRead: : number of pages to read from block
Return values	<ul style="list-style-type: none"> HAL status

36.2.15 HAL_NAND_Write_Page

Function Name	HAL_StatusTypeDef HAL_NAND_Write_Page (NAND_HandleTypeDef * hndl, 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"> hndl: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. pAddress: : pointer to NAND address structure pBuffer: : pointer to source buffer to write NumPageToWrite: : number of pages to write to block
Return values	<ul style="list-style-type: none"> HAL status

36.2.16 HAL_NAND_Read_SpareArea

Function Name	HAL_StatusTypeDef HAL_NAND_Read_SpareArea (NAND_HandleTypeDef * hndl, 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"> hndl: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. pAddress: : pointer to NAND address structure pBuffer: pointer to source buffer to write NumSpareAreaToRead: Number of spare area to read
Return values	<ul style="list-style-type: none"> HAL status

36.2.17 HAL_NAND_Write_SpareArea

Function Name	HAL_StatusTypeDef HAL_NAND_Write_SpareArea (NAND_HandleTypeDef * hndl, 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"> hndl: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. pAddress: : pointer to NAND address structure pBuffer: : pointer to source buffer to write NumSpareAreaTowrite: : number of spare areas to write to

block

Return values

- HAL status

36.2.18 HAL_NAND_Erase_Block

Function Name **HAL_StatusTypeDef HAL_NAND_Erase_Block
(NAND_HandleTypeDef * hhand, NAND_AddressTypeDef *
pAddress)**

Function Description NAND memory Block erase.

Parameters

- **hhnand:** 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 *
hhnand)**

Function Description NAND memory read status.

Parameters

- **hhnand:** 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 *
hhnand, NAND_AddressTypeDef * pAddress)**

Function Description Increment the NAND memory address.

Parameters

- **hhnand:** 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 * hhand)**

Function Description Enables dynamically NAND ECC feature.

Parameters

- **hhnand:** 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	HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hndl)
Function Description	Disables dynamically FMC_NAND ECC feature.
Parameters	<ul style="list-style-type: none"> • hndl: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	<ul style="list-style-type: none"> • HAL status

36.2.23 HAL_NAND_GetECC

Function Name	HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hndl, uint32_t * ECCval, uint32_t Timeout)
Function Description	Disables dynamically NAND ECC feature.
Parameters	<ul style="list-style-type: none"> • hndl: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module. • ECCval: pointer to ECC value • Timeout: maximum timeout to wait
Return values	<ul style="list-style-type: none"> • HAL status

36.2.24 HAL_NAND_GetState

Function Name	HAL_NAND_StateTypeDef HAL_NAND_GetState (NAND_HandleTypeDef * hndl)
Function Description	return the NAND state
Parameters	<ul style="list-style-type: none"> • hndl: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	<ul style="list-style-type: none"> • HAL state

36.2.25 HAL_NAND_Read_Status

Function Name	uint32_t HAL_NAND_Read_Status (NAND_HandleTypeDef * hndl)
Function Description	NAND memory read status.
Parameters	<ul style="list-style-type: none"> • hndl: pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
Return values	<ul style="list-style-type: none"> • NAND status

36.3 NAND Firmware driver defines

36.3.1 NAND

NAND Exported Macros

_HAL_NAND_RESET_HANDLE_STATE **Description:**

- Reset NAND handle state.

Parameters:

- **_HANDLE_**: specifies the NAND

handle.

Return value:

- None

NAND Private Constants

NAND_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: <ul style="list-style-type: none">• NAND memory address computation. Parameters: <ul style="list-style-type: none">• __ADDRESS__: NAND memory address.• __HANDLE__: NAND handle. Return value: <ul style="list-style-type: none">• NAND: Raw address value
ADDR_1ST_CYCLE	Description: <ul style="list-style-type: none">• 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_CFITypeDef

Data Fields

- *uint16_t CFI_1*
- *uint16_t CFI_2*
- *uint16_t CFI_3*
- *uint16_t CFI_4*

Field Documentation

- *uint16_t NOR_CFITypeDef::CFI_1*
< Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- *uint16_t NOR_CFITypeDef::CFI_2*
- *uint16_t NOR_CFITypeDef::CFI_3*
- *uint16_t NOR_CFITypeDef::CFI_4*

37.1.3 NOR_HandleTypeDefDef

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_DelInit\(\)*](#)
- [*HAL_NOR_MspInit\(\)*](#)
- [*HAL_NOR_MspDelInit\(\)*](#)
- [*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"> hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. Timing: pointer to NOR control timing structure ExtTiming: pointer to NOR extended mode timing structure
Return values	<ul style="list-style-type: none"> HAL status

37.2.7 HAL_NOR_DeInit

Function Name	HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)
Function Description	Perform NOR memory De-Initialization sequence.
Parameters	<ul style="list-style-type: none"> hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> HAL status

37.2.8 HAL_NOR_MspInit

Function Name	void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)
Function Description	NOR MSP Init.
Parameters	<ul style="list-style-type: none"> hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> None

37.2.9 HAL_NOR_MspDeInit

Function Name	void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)
Function Description	NOR MSP DeInit.
Parameters	<ul style="list-style-type: none"> hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> None

37.2.10 HAL_NOR_MspWait

Function Name	void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)
Function Description	NOR MSP Wait for Ready/Busy signal.
Parameters	<ul style="list-style-type: none"> hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. Timeout: Maximum timeout value
Return values	<ul style="list-style-type: none"> None

37.2.11 HAL_NOR_Read_ID

Function Name	HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)
Function Description	Read NOR flash IDs.

Parameters	<ul style="list-style-type: none"> hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. pNOR_ID: : pointer to NOR ID structure
Return values	<ul style="list-style-type: none"> HAL status

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	<ul style="list-style-type: none"> hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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.
- Return values HAL status

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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • Address: : Device address
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

37.2.20 HAL_NOR_WriteOperation_Enable

Function Name	HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)
Function Description	Enables dynamically NOR write operation.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> • HAL status

37.2.21 HAL_NOR_WriteOperation_Disable

Function Name	HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)
Function Description	Disables dynamically NOR write operation.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> • HAL status

37.2.22 HAL_NOR_GetState

Function Name	HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)
Function Description	return the NOR controller state
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> • NOR controller state

37.2.23 HAL_NOR_GetStatus

Function Name	HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)
Function Description	Returns the NOR operation status.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • Address: Device address • Timeout: NOR programming Timeout
Return values	<ul style="list-style-type: none"> • NOR_Status The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT

37.3 NOR Firmware driver defines

37.3.1 NOR

NOR Exported Macros

<code>__HAL_NOR_RESET_HANDLE_STATE</code>	Description: <ul style="list-style-type: none">• Reset NOR handle state. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: specifies the NOR handle. Return value: <ul style="list-style-type: none">• 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_ADDRESS1``NOR_MEMORY_ADDRESS2``NOR_MEMORY_ADDRESS3``NOR_MEMORY_ADDRESS4`***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: <ul style="list-style-type: none">• NOR memory address shifting. Parameters: <ul style="list-style-type: none">• __NOR_ADDRESS__: NOR base address• __NOR_MEMORY_WIDTH__: NOR memory width• __ADDRESS__: NOR memory address Return value: <ul style="list-style-type: none">• NOR: shifted address value |
| NOR_WRITE | Description: <ul style="list-style-type: none">• NOR memory write data to specified address. Parameters: <ul style="list-style-type: none">• __ADDRESS__: NOR memory address• __DATA__: Data to write Return value: <ul style="list-style-type: none">• 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_DelInit\(\)*](#)
- [*HAL_PCD_MspInit\(\)*](#)
- [*HAL_PCD_MspDelInit\(\)*](#)

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_ActivateRemoteWakeups()`
- `HAL_PCD_DeActivateRemoteWakeups()`

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	<code>HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)</code>
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">• <code>hpcd</code>: PCD handle
Return values	<ul style="list-style-type: none">• HAL status

38.2.7 HAL_PCD_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_PCD_DelInit (PCD_HandleTypeDef * hpcd)</code>
Function Description	DeInitializes the PCD peripheral.
Parameters	<ul style="list-style-type: none">• <code>hpcd</code>: PCD handle
Return values	<ul style="list-style-type: none">• HAL status

38.2.8 HAL_PCD_MspInit

Function Name	<code>void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)</code>
Function Description	Initializes the PCD MSP.
Parameters	<ul style="list-style-type: none">• <code>hpcd</code>: PCD handle
Return values	<ul style="list-style-type: none">• None

38.2.9 HAL_PCD_MspDelInit

Function Name	<code>void HAL_PCD_MspDelInit (PCD_HandleTypeDef * hpcd)</code>
---------------	---

	Function Description	Deinitializes PCD MSP.
	Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
	Return values	<ul style="list-style-type: none"> • None
38.2.10 HAL_PCD_Start		
	Function Name	HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)
	Function Description	Start The USB OTG Device.
	Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
	Return values	<ul style="list-style-type: none"> • HAL status
38.2.11 HAL_PCD_Stop		
	Function Name	HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)
	Function Description	Stop The USB OTG Device.
	Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
	Return values	<ul style="list-style-type: none"> • HAL status
38.2.12 HAL_PCD_IRQHandler		
	Function Name	void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)
	Function Description	This function handles PCD interrupt request.
	Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
	Return values	<ul style="list-style-type: none"> • HAL status
38.2.13 HAL_PCD_DataOutStageCallback		
	Function Name	void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
	Function Description	Data out stage callbacks.
	Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • epnum: endpoint number
	Return values	<ul style="list-style-type: none"> • None
38.2.14 HAL_PCD_DataInStageCallback		
	Function Name	void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
	Function Description	Data IN stage callbacks.
	Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • epnum: endpoint number
	Return values	<ul style="list-style-type: none"> • None

38.2.15 HAL_PCD_SetupStageCallback

Function Name	<code>void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)</code>
Function Description	Setup stage callback.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None

38.2.16 HAL_PCD_SOFCallback

Function Name	<code>void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)</code>
Function Description	USB Start Of Frame callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None

38.2.17 HAL_PCD_ResetCallback

Function Name	<code>void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)</code>
Function Description	USB Reset callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None

38.2.18 HAL_PCD_SuspendCallback

Function Name	<code>void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)</code>
Function Description	Suspend event callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None

38.2.19 HAL_PCD_ResumeCallback

Function Name	<code>void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)</code>
Function Description	Resume event callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None

38.2.20 HAL_PCD_ISOOUTIncompleteCallback

Function Name	<code>void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epi)</code>
Function Description	Incomplete ISO OUT callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• epi: endpoint number
Return values	<ul style="list-style-type: none">• None

Return values	<ul style="list-style-type: none"> None
38.2.21 HAL_PCD_ISOINIncompleteCallback	
Function Name	void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epcnum)
Function Description	Incomplete ISO IN callbacks.
Parameters	<ul style="list-style-type: none"> hpcd: PCD handle epcnum: endpoint number
Return values	<ul style="list-style-type: none"> None
38.2.22 HAL_PCD_ConnectCallback	
Function Name	void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)
Function Description	Connection event callbacks.
Parameters	<ul style="list-style-type: none"> hpcd: PCD handle
Return values	<ul style="list-style-type: none"> None
38.2.23 HAL_PCD_DisconnectCallback	
Function Name	void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)
Function Description	Disconnection event callbacks.
Parameters	<ul style="list-style-type: none"> hpcd: PCD handle
Return values	<ul style="list-style-type: none"> None
38.2.24 HAL_PCD_DevConnect	
Function Name	HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)
Function Description	Connect the USB device.
Parameters	<ul style="list-style-type: none"> hpcd: PCD handle
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hpcd: PCD handle
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> • hpcd: PCD handle • address: new device address
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address • ep_mps: endpoint max packet size • ep_type: endpoint type
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address • pBuf: pointer to the reception buffer • len: amount of data to be received
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address

Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address • pBuf: pointer to the transmission buffer • len: amount of data to be sent
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address
Return values	<ul style="list-style-type: none"> • 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"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • HAL status

38.2.36 HAL_PCD_DeActivateRemoteWakeUp

Function Name	HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeUp (PCD_HandleTypeDef * hpcd)
Function Description	HAL_PCD_DeActivateRemoteWakeUp : de-active remote wake-up signalling.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • HAL status

38.2.37 HAL_PCD_GetState

Function Name	PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)
Function Description	Return the PCD state.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • HAL state

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

	d to the USB HS EXTI Line
USB_OTG_FS_WAKEUP_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 PCDEEx 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"> • hpcd: PCD handle • fifo: The number of Tx fifo • size: Fifo size
Return values	<ul style="list-style-type: none"> • HAL status

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"> • hpcd: PCD handle • size: Size of Rx fifo
Return values	<ul style="list-style-type: none"> • HAL status

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"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • HAL status

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 • **hpcd:** PCD handle

Return values • HAL status

39.1.6 HAL_PCDEx_LPM_Callback

Function Name **void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef *
hpcd, PCD_LPM_MsgTypeDef msg)**

Function Description HAL_PCDEx_LPM_Callback : Send LPM message to user layer.

Parameters • **hpcd:** PCD handle
 • **msg:** LPM message

Return values • HAL status

40 HAL PWR Generic Driver

40.1 PWR Firmware driver registers structures

40.1.1 PWR_PVDTTypeDef

Data Fields

- *uint32_t PVDLevel*
- *uint32_t Mode*

Field Documentation

- *uint32_t PWR_PVDTTypeDef::PVDLevel*
PVDLevel: Specifies the PVD detection level. This parameter can be a value of [PWR_PVD_detection_level](#)
- *uint32_t PWR_PVDTTypeDef::Mode*
Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [PWR_PVD_Mode](#)

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_EnableSEVOOnPend()`
- `HAL_PWR_DisableSEVOOnPend()`

40.2.3 HAL_PWR_Delinit

Function Name	<code>void HAL_PWR_Delinit (void)</code>
Function Description	Deinitializes the HAL PWR peripheral registers to their default reset values.
Return values	<ul style="list-style-type: none"> • None

40.2.4 HAL_PWR_EnableBkUpAccess

Function Name	<code>void HAL_PWR_EnableBkUpAccess (void)</code>
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"> • None

Notes	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> 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_PVDTTypeDef * sConfigPVD)
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> sConfigPVD: pointer to an PWR_PVDTTypeDef structure that contains the configuration information for the PVD.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> None

40.2.8 HAL_PWR_DisablePVD

Function Name	void HAL_PWR_DisablePVD (void)
Function Description	Disables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> None

40.2.9 HAL_PWR_EnableWakeUpPin

Function Name	void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinPolarity)
Function Description	Enable the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> 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_LOW PWR_WAKEUP_PIN2_HIGH or PWR_WAKEUP_PIN2_LOW PWR_WAKEUP_PIN3_HIGH or PWR_WAKEUP_PIN3_LOW PWR_WAKEUP_PIN4_HIGH or PWR_WAKEUP_PIN4_LOW PWR_WAKEUP_PIN5_HIGH or PWR_WAKEUP_PIN5_LOW PWR_WAKEUP_PIN6_HIGH or PWR_WAKEUP_PIN6_LOW
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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 instruction PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> 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	<code>void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)</code>
Function Description	Enters Stop mode.
Parameters	<ul style="list-style-type: none"> • Regulator: Specifies the regulator state in Stop mode. This parameter can be one of the following values: PWR_MAINREGULATOR_ON: Stop mode with regulator ON PWR_LOWPOWERREGULATOR_ON: Stop mode with low power regulator ON • STOPEntry: Specifies if Stop mode 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
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • In Stop mode, all I/O pins keep the same state as in Run mode. • When exiting Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock. • When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

40.2.13 HAL_PWR_EnterSTANDBYMode

Function Name	<code>void HAL_PWR_EnterSTANDBYMode (void)</code>
Function Description	Enters Standby mode.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • 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.

40.2.14 HAL_PWR_PVD_IRQHandler

Function Name	<code>void HAL_PWR_PVD_IRQHandler (void)</code>
Function Description	This function handles the PWR PVD interrupt request.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This API should be called under the PVD_IRQHandler().

40.2.15 HAL_PWR_PVDCallback

Function Name	void HAL_PWR_PVDCallback(void)
Function Description	PWR PVD interrupt callback.
Return values	<ul style="list-style-type: none"> None

40.2.16 HAL_PWR_EnableSleepOnExit

Function Name	void HAL_PWR_EnableSleepOnExit(void)
Function Description	Indicates Sleep-On-Exit when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> 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.

40.2.17 HAL_PWR_DisableSleepOnExit

Function Name	void HAL_PWR_DisableSleepOnExit(void)
Function Description	Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

40.2.18 HAL_PWR_EnableSEVOnPend

Function Name	void HAL_PWR_EnableSEVOnPend(void)
Function Description	Enables CORTEX M4 SEVONPEND bit.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> 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.

40.2.19 HAL_PWR_DisableSEVOnPend

Function Name	void HAL_PWR_DisableSEVOnPend(void)
Function Description	Disables CORTEX M4 SEVONPEND bit.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> 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.

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

`_HAL_PWR_GET_FLAG`

Description:

- Check PWR flag is set or not.

Parameters:

- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
 - `PWR_FLAG_WU`: Wake Up flag. This flag indicates that a wakeup event was received 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

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).

`__HAL_PWR_CLEAR_FLAG`

Description:

- Clear the PWR's pending flags.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - PWR_FLAG_SB: StandBy flag

`__HAL_PWR_PVD_EXTI_ENABLE_IT`

Description:

- Enable the PVD EXTI Line 16.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_IT`

Description:

- Disable the PVD EXTI Line 16.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_ENABLE_EVENT`

Description:

- Enable event on PVD EXTI Line 16.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_DISABLE_EVENT`

Description:

- Disable event on PVD EXTI Line 16.

Return value:

- None.

`__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE`

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

PWR_PVD_MODE_IT_RISING_FALLING	edge trigger detection
PWR_PVD_MODE_EVENT_RISING	External Interrupt Mode with Rising/Falling edge trigger detection
PWR_PVD_MODE_EVENT_FALLING	Event Mode with Rising edge trigger detection
PWR_PVD_MODE_EVENT_RISING_FALLING	Event Mode with Falling edge trigger detection
	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	<code>HAL_StatusTypeDef HAL_PWREx_EnableBkUpReg (void)</code>
Function Description	Enables the Backup Regulator.
Return values	<ul style="list-style-type: none"> • HAL status

41.1.3 `HAL_PWREx_DisableBkUpReg`

Function Name	<code>HAL_StatusTypeDef HAL_PWREx_DisableBkUpReg (void)</code>
Function Description	Disables the Backup Regulator.
Return values	<ul style="list-style-type: none"> • HAL status

41.1.4 `HAL_PWREx_EnableFlashPowerDown`

Function Name	<code>void HAL_PWREx_EnableFlashPowerDown (void)</code>
Function Description	Enables the Flash Power Down in Stop mode.
Return values	<ul style="list-style-type: none"> • None

41.1.5 `HAL_PWREx_DisableFlashPowerDown`

Function Name	<code>void HAL_PWREx_DisableFlashPowerDown (void)</code>
Function Description	Disables the Flash Power Down in Stop mode.
Return values	<ul style="list-style-type: none"> • None

41.1.6 `HAL_PWREx_EnableMainRegulatorLowVoltage`

Function Name	void HAL_PWREx_EnableMainRegulatorLowVoltage (void)
Function Description	Enables Main Regulator low voltage mode.
Return values	<ul style="list-style-type: none"> • None

41.1.7 **HAL_PWREx_DisableMainRegulatorLowVoltage**

Function Name	void HAL_PWREx_DisableMainRegulatorLowVoltage (void)
Function Description	Disables Main Regulator low voltage mode.
Return values	<ul style="list-style-type: none"> • None

41.1.8 **HAL_PWREx_EnableLowRegulatorLowVoltage**

Function Name	void HAL_PWREx_EnableLowRegulatorLowVoltage (void)
Function Description	Enables Low Power Regulator low voltage mode.
Return values	<ul style="list-style-type: none"> • None

41.1.9 **HAL_PWREx_DisableLowRegulatorLowVoltage**

Function Name	void HAL_PWREx_DisableLowRegulatorLowVoltage (void)
Function Description	Disables Low Power Regulator low voltage mode.
Return values	<ul style="list-style-type: none"> • None

41.1.10 **HAL_PWREx_EnableOverDrive**

Function Name	HAL_StatusTypeDef HAL_PWREx_EnableOverDrive (void)
Function Description	Activates the Over-Drive mode.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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). • 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.

41.1.11 **HAL_PWREx_DisableOverDrive**

Function Name	HAL_StatusTypeDef HAL_PWREx_DisableOverDrive (void)
Function Description	Deactivates the Over-Drive mode.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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). • 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.

41.1.12 HAL_PWREx_EnterUnderDriveSTOPMode

Function Name	<code>HAL_StatusTypeDef HAL_PWREx_EnterUnderDriveSTOPMode (uint32_t Regulator, uint8_t STOPEntry)</code>
Function Description	Enters in Under-Drive STOP mode.
Parameters	<ul style="list-style-type: none"> • Regulator: 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 modePWR_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 • STOPEntry: 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 instructionPWR_SLEEPENTRY_WFE: enter STOP mode with WFE instruction
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This mode can be selected only when the Under-Drive is already active • 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. • In Stop mode, all I/O pins keep the same state as in Run mode. • When exiting Stop mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock. • When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

41.1.13 HAL_PWREx_GetVoltageRange

Function Name	<code>uint32_t HAL_PWREx_GetVoltageRange (void)</code>
Function Description	Returns Voltage Scaling Range.

Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL Status
Notes	<ul style="list-style-type: none"> To update the system clock frequency(SYCLK): 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_ENA
BLE
__HAL_PWR_OVERDRIVESWITCHING_DISA
BLE

```



[__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_UDRVDRDY](#): 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](#):

- Wakeup Pin Flag for PC13
- PWR_WAKEUP_PIN_FLAG5: Wakeup Pin Flag for PI8
- PWR_WAKEUP_PIN_FLAG6: Wakeup Pin Flag for PI11

_HAL_PWR_CLEAR_WAKEUP_FLAG**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_Underdrive_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_HandleTypeDef 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_DelInit()*
- *HAL_QSPI_MspInit()*
- *HAL_QSPI_MspDelInit()*

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	<code>HAL_StatusTypeDef HAL_QSPI_Init (QSPI_HandleTypeDef * hspi)</code>
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"> • <code>hspi</code>: qspi handle
Return values	<ul style="list-style-type: none"> • HAL status

42.2.6 `HAL_QSPI_DelInit`

Function Name	<code>HAL_StatusTypeDef HAL_QSPI_DelInit (QSPI_HandleTypeDef * hspi)</code>
Function Description	Delinitializes the QSPI peripheral.
Parameters	<ul style="list-style-type: none"> • <code>hspi</code>: qspi handle
Return values	<ul style="list-style-type: none"> • HAL status

42.2.7 `HAL_QSPI_MspInit`

Function Name	<code>void HAL_QSPI_MspInit (QSPI_HandleTypeDef * hspi)</code>
Function Description	QSPI MSP Init.
Parameters	<ul style="list-style-type: none"> • <code>hspi</code>: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.8 `HAL_QSPI_MspDelInit`

Function Name	<code>void HAL_QSPI_MspDelInit (QSPI_HandleTypeDef * hspi)</code>
Function Description	QSPI MSP Delinit.
Parameters	<ul style="list-style-type: none"> • <code>hspi</code>: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.9 `HAL_QSPI_IRQHandler`

Function Name	void HAL_QSPI_IRQHandler (QSPI_HandleTypeDef * hspi)
Function Description	This function handles QSPI interrupt request.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None.

42.2.10 HAL_QSPI_Command

Function Name	HAL_StatusTypeDef HAL_QSPI_Command (QSPI_HandleTypeDef * hspi, QSPI_CommandTypeDef * cmd, uint32_t Timeout)
Function Description	Sets the command configuration.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle • cmd: : structure that contains the command configuration information • Timeout: : Time out duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used only in Indirect Read or Write Modes

42.2.11 HAL_QSPI_Command_IT

Function Name	HAL_StatusTypeDef HAL_QSPI_Command_IT (QSPI_HandleTypeDef * hspi, QSPI_CommandTypeDef * cmd)
Function Description	Sets the command configuration in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle • cmd: : structure that contains the command configuration information
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used only in Indirect Read or Write Modes

42.2.12 HAL_QSPI_Transmit

Function Name	HAL_StatusTypeDef HAL_QSPI_Transmit (QSPI_HandleTypeDef * hspi, uint8_t * pData, uint32_t Timeout)
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle • pData: pointer to data buffer • Timeout: : Time out duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used only in Indirect Write Mode

42.2.13 HAL_QSPI_Receive

Function Name	HAL_StatusTypeDef HAL_QSPI_Receive (QSPI_HandleTypeDef * hspi, uint8_t * pData, uint32_t
---------------	---

Timeout)

Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle • pData: pointer to data buffer • Timeout: : Time out duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used only in Indirect Read Mode

42.2.14 HAL_QSPI_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_QSPI_Transmit_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle • pData: pointer to data buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used only in Indirect Write Mode

42.2.15 HAL_QSPI_Receive_IT

Function Name	HAL_StatusTypeDef HAL_QSPI_Receive_IT (QSPI_HandleTypeDef * hqspi, uint8_t * pData)
Function Description	Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle • pData: pointer to data buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used only in Indirect Read Mode

42.2.16 HAL_QSPI_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_QSPI_Transmit_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)
Function Description	Sends an amount of data in non blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle • pData: pointer to data buffer
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used only in Indirect Write Mode

42.2.17 HAL_QSPI_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_QSPI_Receive_DMA (QSPI_HandleTypeDef * hqspi, uint8_t * pData)
Function Description	Receives an amount of data in non blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle

- **pData:** pointer to data buffer.
- Return values HAL status
- Notes This function is used only in Indirect Read Mode

42.2.18 HAL_QSPI_AutoPolling

Function Name	HAL_StatusTypeDef HAL_QSPI_AutoPolling (QSPI_HandleTypeDef * hspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)
Function Description	Configure the QSPI Automatic Polling Mode in blocking mode.
Parameters	<ul style="list-style-type: none"> • hspi: 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 * hspi, QSPI_CommandTypeDef * cmd, QSPI_AutoPollingTypeDef * cfg)
Function Description	Configure the QSPI Automatic Polling Mode in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • hspi: 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 * hspi, QSPI_CommandTypeDef * cmd, QSPI_MemoryMappedTypeDef * cfg)
Function Description	Configure the Memory Mapped mode.
Parameters	<ul style="list-style-type: none"> • hspi: 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	void HAL_QSPI_ErrorCallback (QSPI_HandleTypeDef * hspi)
Function Description	Transfer Error callbacks.
Parameters	<ul style="list-style-type: none">• hspi: QSPI handle
Return values	<ul style="list-style-type: none">• None

42.2.22 HAL_QSPI_CmdCpltCallback

Function Name	void HAL_QSPI_CmdCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Command completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: QSPI handle
Return values	<ul style="list-style-type: none">• None

42.2.23 HAL_QSPI_RxCpltCallback

Function Name	void HAL_QSPI_RxCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: QSPI handle
Return values	<ul style="list-style-type: none">• None

42.2.24 HAL_QSPI_TxCpltCallback

Function Name	void HAL_QSPI_TxCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: QSPI handle
Return values	<ul style="list-style-type: none">• None

42.2.25 HAL_QSPI_RxHalfCpltCallback

Function Name	void HAL_QSPI_RxHalfCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: QSPI handle
Return values	<ul style="list-style-type: none">• None

42.2.26 HAL_QSPI_TxHalfCpltCallback

Function Name	void HAL_QSPI_TxHalfCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: QSPI handle

Return values	<ul style="list-style-type: none">None
---------------	--

42.2.27 HAL_QSPI_FifoThresholdCallback

Function Name	void HAL_QSPI_FifoThresholdCallback (QSPI_HandleTypeDefDef * hqspi)
Function Description	FIFO Threshold callbacks.
Parameters	<ul style="list-style-type: none">hqspi: QSPI handle
Return values	<ul style="list-style-type: none">None

42.2.28 HAL_QSPI_StatusMatchCallback

Function Name	void HAL_QSPI_StatusMatchCallback (QSPI_HandleTypeDefDef * hqspi)
Function Description	Status Match callbacks.
Parameters	<ul style="list-style-type: none">hqspi: QSPI handle
Return values	<ul style="list-style-type: none">None

42.2.29 HAL_QSPI_TimeOutCallback

Function Name	void HAL_QSPI_TimeOutCallback (QSPI_HandleTypeDefDef * hqspi)
Function Description	Timeout callbacks.
Parameters	<ul style="list-style-type: none">hqspi: QSPI handle
Return values	<ul style="list-style-type: none">None

42.2.30 HAL_QSPI_GetState

Function Name	HAL_QSPI_StateTypeDef HAL_QSPI_GetState (QSPI_HandleTypeDefDef * hqspi)
Function Description	Return the QSPI state.
Parameters	<ul style="list-style-type: none">hqspi: QSPI handle
Return values	<ul style="list-style-type: none">HAL state

42.2.31 HAL_QSPI_GetError

Function Name	uint32_t HAL_QSPI_GetError (QSPI_HandleTypeDefDef * hqspi)
Function Description	Return the QSPI error code.
Parameters	<ul style="list-style-type: none">hqspi: QSPI handle
Return values	<ul style="list-style-type: none">QSPI Error Code

42.2.32 HAL_QSPI_Abort

Function Name	HAL_StatusTypeDef HAL_QSPI_Abort (QSPI_HandleTypeDefDef * hqspi)
---------------	---

Function Description	Abort the current transmission.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle
Return values	<ul style="list-style-type: none"> • HAL status

42.2.33 HAL_QSPI_SetTimeout

Function Name	void HAL_QSPI_SetTimeout (QSPI_HandleTypeDef * hspi, uint32_t Timeout)
Function Description	Set QSPI timeout.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle. • Timeout: Timeout for the QSPI memory access.
Return values	<ul style="list-style-type: none"> • None

42.2.34 HAL_QSPI_ErrorCallback

Function Name	void HAL_QSPI_ErrorCallback (QSPI_HandleTypeDef * hspi)
Function Description	Transfer Error callbacks.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.35 HAL_QSPI_FifoThresholdCallback

Function Name	void HAL_QSPI_FifoThresholdCallback (QSPI_HandleTypeDef * hspi)
Function Description	FIFO Threshold callbacks.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.36 HAL_QSPI_CmdCpltCallback

Function Name	void HAL_QSPI_CmdCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Command completed callbacks.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.37 HAL_QSPI_RxCpltCallback

Function Name	void HAL_QSPI_RxCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hqspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.38 HAL_QSPI_TxCpltCallback



Function Name	void HAL_QSPI_TxCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.39 HAL_QSPI_RxHalfCpltCallback

Function Name	void HAL_QSPI_RxHalfCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.40 HAL_QSPI_TxHalfCpltCallback

Function Name	void HAL_QSPI_TxHalfCpltCallback (QSPI_HandleTypeDef * hspi)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.41 HAL_QSPI_StatusMatchCallback

Function Name	void HAL_QSPI_StatusMatchCallback (QSPI_HandleTypeDef * hspi)
Function Description	Status Match callbacks.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.42 HAL_QSPI_TimeOutCallback

Function Name	void HAL_QSPI_TimeOutCallback (QSPI_HandleTypeDef * hspi)
Function Description	Timeout callbacks.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle
Return values	<ul style="list-style-type: none"> • None

42.2.43 HAL_QSPI_GetState

Function Name	HAL_QSPI_StateTypeDef HAL_QSPI_GetState (QSPI_HandleTypeDef * hspi)
Function Description	Return the QSPI state.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle

Return values	<ul style="list-style-type: none"> • HAL state
---------------	---

42.2.44 HAL_QSPI_GetError

Function Name	<code>uint32_t HAL_QSPI_GetError (QSPI_HandleTypeDef * hspi)</code>
Function Description	Return the QSPI error code.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle
Return values	<ul style="list-style-type: none"> • QSPI Error Code

42.2.45 HAL_QSPI_Abort

Function Name	<code>HAL_StatusTypeDef HAL_QSPI_Abort (QSPI_HandleTypeDef * hspi)</code>
Function Description	Abort the current transmission.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle
Return values	<ul style="list-style-type: none"> • HAL status

42.2.46 HAL_QSPI_SetTimeout

Function Name	<code>void HAL_QSPI_SetTimeout (QSPI_HandleTypeDef * hspi, uint32_t Timeout)</code>
Function Description	Set QSPI timeout.
Parameters	<ul style="list-style-type: none"> • hspi: QSPI handle. • Timeout: Timeout for the QSPI memory access.
Return values	<ul style="list-style-type: none"> • None

42.3 QSPI Firmware driver defines

42.3.1 QSPI

QSPI Address Mode

<code>QSPI_ADDRESS_NONE</code>	No address
<code>QSPI_ADDRESS_1_LINE</code>	Address on a single line
<code>QSPI_ADDRESS_2_LINES</code>	Address on two lines
<code>QSPI_ADDRESS_4_LINES</code>	Address on four lines

QSPI Address Size

<code>QSPI_ADDRESS_8_BITS</code>	8-bit address
<code>QSPI_ADDRESS_16_BITS</code>	16-bit address
<code>QSPI_ADDRESS_24_BITS</code>	24-bit address
<code>QSPI_ADDRESS_32_BITS</code>	32-bit address

QSPI Alternate Bytes Mode

<code>QSPI_ALTERNATE_BYTES_NONE</code>	No alternate bytes
<code>QSPI_ALTERNATE_BYTES_1_LINE</code>	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` **Description:**

- Reset QSPI handle state.

Parameters:

- `_HANDLE_`: QSPI handle.

Return value:

- None

`_HAL_QSPI_ENABLE`

Description:

- Enable QSPI.

Parameters:

- `_HANDLE_`: specifies the QSPI Handle.

Return value:

- None

`_HAL_QSPI_DISABLE`

Description:

- Disable QSPI.

Parameters:

- `_HANDLE_`: specifies the QSPI Handle.

Return value:

- None

`_HAL_QSPI_ENABLE_IT`

Description:

- 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

_HAL_QSPI_DISABLE_IT

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

_HAL_QSPI_GET_IT_SOURCE

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

- 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).

[__HAL_QSPI_GET_FLAG](#)**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

[__HAL_QSPI_CLEAR_FLAG](#)**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

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 HSISState*
- *uint32_t HSICalibrationValue*
- *uint32_t LSISState*
- *RCC_PLLInitTypeDef PLL*

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
The oscillators to be configured. This parameter can be a value of [**RCC_Oscillator_Type**](#)
- ***uint32_t RCC_OscInitTypeDef::HSEState***
The new state of the HSE. This parameter can be a value of [**RCC_HSE_Config**](#)
- ***uint32_t RCC_OscInitTypeDef::LSEState***
The new state of the LSE. This parameter can be a value of [**RCC_LSE_Config**](#)
- ***uint32_t RCC_OscInitTypeDef::HSIState***
The new state of the HSI. This parameter can be a value of [**RCC_HSI_Config**](#)
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
The calibration trimming value (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 (<=48 MHz) and the SDIO (<= 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 to 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	void HAL_RCC_DelInit (void)
Function Description	Resets the RCC clock configuration to the default reset state.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> The default reset state of the clock configuration is given below: HSI ON and used as system clock sourceHSE, PLL and PLLI2S OFFAHB, APB1 and APB2 prescaler set to 1.CSS, MCO1 and MCO2 OFFAll interrupts disabled This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

43.2.6 HAL_RCC_OscConfig

Function Name	HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)
Function Description	Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.
Parameters	<ul style="list-style-type: none"> RCC_OscInitStruct: pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> The PLL is not disabled when used as system clock.

43.2.7 HAL_RCC_ClockConfig

Function Name	HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)
Function Description	Initializes the CPU, AHB and APB busses clocks according to the specified parameters in the RCC_ClkInitStruct.
Parameters	<ul style="list-style-type: none"> RCC_ClkInitStruct: pointer to an RCC_ClkInitTypeDef structure that contains the configuration information for the RCC peripheral. FLatency: FLASH Latency, this parameter depend on device selected
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function The HSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready. You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source. Depending on the device voltage range, the software has to

set correctly HPRE[3:0] bits to ensure that HCLK not exceed the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")

43.2.8 HAL_RCC_MCOConfig

Function Name	<code>void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOsource, uint32_t RCC_MCODiv)</code>
Function Description	Selects the clock source to output on MCO1 pin(PA8) or on MCO2 pin(PC9).
Parameters	<ul style="list-style-type: none"> • RCC_MCOx: 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). • RCC_MCOsource: specifies the clock source to output. This parameter can be one of the following values: RCC_MCO1SOURCE_HSI: HSI clock selected as MCO1 sourceRCC_MCO1SOURCE_LSE: LSE clock selected as MCO1 sourceRCC_MCO1SOURCE_HSE: HSE clock selected as MCO1 sourceRCC_MCO1SOURCE_PLLCLK: main PLL clock selected as MCO1 sourceRCC_MCO2SOURCE_SYSCLK: System clock (SYSCLK) selected as MCO2 sourceRCC_MCO2SOURCE_PLLI2SCLK: PLLI2S clock selected as MCO2 sourceRCC_MCO2SOURCE_HSE: HSE clock selected as MCO2 sourceRCC_MCO2SOURCE_PLLCLK: main PLL clock selected as MCO2 source • RCC_MCODiv: specifies the MCOx prescaler. This parameter can be one of the following values: RCC_MCODIV_1: no division applied to MCOx clockRCC_MCODIV_2: division by 2 applied to MCOx clockRCC_MCODIV_3: division by 3 applied to MCOx clockRCC_MCODIV_4: division by 4 applied to MCOx clockRCC_MCODIV_5: division by 5 applied to MCOx clock
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • PA8/PC9 should be configured in alternate function mode.

43.2.9 HAL_RCC_EnableCSS

Function Name	<code>void HAL_RCC_EnableCSS (void)</code>
Function Description	Enables the Clock Security System.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • 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.

43.2.10 HAL_RCC_DisableCSS

Function Name	void HAL_RCC_DisableCSS (void)
Function Description	Disables the Clock Security System.
Return values	<ul style="list-style-type: none"> • None

43.2.11 HAL_RCC_GetSysClockFreq

Function Name	uint32_t HAL_RCC_GetSysClockFreq (void)
Function Description	Returns the SYSCLK frequency.
Return values	<ul style="list-style-type: none"> • SYSCLK frequency
Notes	<ul style="list-style-type: none"> • The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source: • If SYSCLK source is HSI, function returns values based on HSI_VALUE(*) • If SYSCLK source is HSE, function returns values based on HSE_VALUE(**) • If SYSCLK source is PLL, function returns values based on HSE_VALUE(**) or HSI_VALUE(*) multiplied/divided by the PLL factors. • (*) HSI_VALUE is a constant defined in stm32f7xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature. • (**) HSE_VALUE is a constant defined in 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. • The result of this function could be not correct when using fractional value for HSE crystal. • This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters. • Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

43.2.12 HAL_RCC_GetHCLKFreq

Function Name	uint32_t HAL_RCC_GetHCLKFreq (void)
Function Description	Returns the HCLK frequency.
Return values	<ul style="list-style-type: none"> • HCLK frequency
Notes	<ul style="list-style-type: none"> • Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect. • The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

43.2.13 HAL_RCC_GetPCLK1Freq

Function Name	<code>uint32_t HAL_RCC_GetPCLK1Freq (void)</code>
Function Description	Returns the PCLK1 frequency.
Return values	<ul style="list-style-type: none"> PCLK1 frequency
Notes	<ul style="list-style-type: none"> 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.

43.2.14 HAL_RCC_GetPCLK2Freq

Function Name	<code>uint32_t HAL_RCC_GetPCLK2Freq (void)</code>
Function Description	Returns the PCLK2 frequency.
Return values	<ul style="list-style-type: none"> PCLK2 frequency
Notes	<ul style="list-style-type: none"> 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.

43.2.15 HAL_RCC_GetOscConfig

Function Name	<code>void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)</code>
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> RCC_OscInitStruct: pointer to an RCC_OscInitTypeDef structure that will be configured.
Return values	<ul style="list-style-type: none"> None

43.2.16 HAL_RCC_GetClockConfig

Function Name	<code>void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)</code>
Function Description	Configures the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> RCC_ClkInitStruct: pointer to an RCC_ClkInitTypeDef structure that will be configured. pFLatency: Pointer on the Flash Latency.
Return values	<ul style="list-style-type: none"> None

43.2.17 HAL_RCC_NMI_IRQHandler

Function Name	<code>void HAL_RCC_NMI_IRQHandler (void)</code>
Function Description	This function handles the RCC CSS interrupt request.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> This API should be called under the NMI_Handler().

43.2.18 HAL_RCC_CSSCallback

Function Name	void HAL_RCC_CSSCallback (void)
Function Description	RCC Clock Security System interrupt callback.
Return values	<ul style="list-style-type: none"> • None

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_CRYPT_IS_CLK_SLEEP_ENABLED
_HAL_RCC_HASH_IS_CLK_SLEEP_ENABLED
_HAL_RCC_CRYPT_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_PLLI2SRDY
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_LPWRST

Flags Interrupts Management

`__HAL_RCC_ENABLE_IT`

Description:

- Enable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to enable the selected interrupts).

Parameters:

- `__INTERRUPT__`: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
 - `RCC_IT_LSIRDY`: LSI ready interrupt.
 - `RCC_IT_LSERDY`: LSE ready interrupt.
 - `RCC_IT_HSIRDY`: HSI ready interrupt.
 - `RCC_IT_HSERDY`: HSE ready interrupt.
 - `RCC_IT_PLLRDY`: Main PLL ready interrupt.
 - `RCC_IT_PLLI2SRDY`: PLLI2S ready interrupt.

`__HAL_RCC_DISABLE_IT`

Description:

- Disable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to disable the selected interrupts).

Parameters:

- `__INTERRUPT__`: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
 - `RCC_IT_LSIRDY`: LSI ready interrupt.

- RCC_IT_LSERDY: LSE ready interrupt.
- RCC_IT_HSIRDY: HSI ready interrupt.
- RCC_IT_HSERDY: HSE ready interrupt.
- RCC_IT_PLLRDY: Main PLL ready interrupt.
- RCC_IT_PLLI2SRDY: PLLI2S ready interrupt.

_HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC_CIR[23:16] bits to clear the selected interrupt pending bits.)

Parameters:

- _INTERRUPT_: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt.
 - RCC_IT_LSERDY: LSE ready interrupt.
 - RCC_IT_HSIRDY: HSI ready interrupt.
 - RCC_IT_HSERDY: HSE ready interrupt.
 - RCC_IT_PLLRDY: Main PLL ready interrupt.
 - RCC_IT_PLLI2SRDY: PLLI2S ready interrupt.
 - RCC_IT_CSS: Clock Security System interrupt

_HAL_RCC_GET_IT

Description:

- Check the RCC's interrupt has occurred or not.

Parameters:

- _INTERRUPT_: specifies the RCC interrupt source to check. This parameter can be one of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt.
 - RCC_IT_LSERDY: LSE ready interrupt.
 - RCC_IT_HSIRDY: HSI ready interrupt.
 - RCC_IT_HSERDY: HSE ready interrupt.
 - RCC_IT_PLLRDY: Main PLL ready interrupt.
 - RCC_IT_PLLI2SRDY: PLLI2S ready interrupt.
 - RCC_IT_CSS: Clock Security System interrupt

Return value:

- The: new state of _INTERRUPT_ (TRUE

or FALSE).

`_HAL_RCC_CLEAR_RESET_FLAGS`

`RCC_FLAG_MASK`

Description:

- Check RCC flag is set or not.

Parameters:

- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
 - `RCC_FLAG_HSIRDY`: HSI oscillator clock ready.
 - `RCC_FLAG_HSERDY`: HSE oscillator clock ready.
 - `RCC_FLAG_PLLRDY`: Main PLL clock ready.
 - `RCC_FLAG_PLLI2SRDY`: PLLI2S clock ready.
 - `RCC_FLAG_LSERDY`: LSE oscillator clock ready.
 - `RCC_FLAG_LSIRDY`: LSI oscillator clock ready.
 - `RCC_FLAG_BORRST`: POR/PDR or BOR reset.
 - `RCC_FLAG_PINRST`: Pin reset.
 - `RCC_FLAG_PORRST`: POR/PDR reset.
 - `RCC_FLAG_SFTRST`: Software reset.
 - `RCC_FLAG_IWDGRST`: Independent Watchdog reset.
 - `RCC_FLAG_WWDGRST`: Window Watchdog reset.
 - `RCC_FLAG_LPWRRST`: Low Power reset.

Return value:

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

`_HAL_RCC_GET_FLAG`

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](#)
RCE

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_OSC_SOURCE](#)
RCE

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_FORC
E``__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.

- PLL_N: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between Min_Data = 192 and Max_Data = 432.
- PLL_P: specifies the division factor for main system clock (SYSCLK) This parameter must be a number in the range {2, 4, 6, or 8}.
- PLL_Q: 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:

- PLL_SOURCE: 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

Notes:

- This function must be used only when the main PLL is disabled.

`_HAL_RCC_PLL_PLLM_CONFIG`**Description:**

- Macro to configure the PLL multiplication factor.

Parameters:

- `_PLL_M_`: 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 RCCEEx 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_PерiphCLKInitTypeDef

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_PерiphCLKInitTypeDef::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
`RCCEEx_UART8_Clock_Source`
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection`**
I2C1 clock source This parameter can be a value of **`RCCEEx_I2C1_Clock_Source`**
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c2ClockSelection`**
I2C2 clock source This parameter can be a value of **`RCCEEx_I2C2_Clock_Source`**
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection`**
I2C3 clock source This parameter can be a value of **`RCCEEx_I2C3_Clock_Source`**
- **`uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection`**
I2C4 clock source This parameter can be a value of **`RCCEEx_I2C4_Clock_Source`**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection`**
Specifies LPTIM1 clock source This parameter can be a value of
`RCCEEx_LPTIM1_Clock_Source`
- **`uint32_t RCC_PeriphCLKInitTypeDef::CecClockSelection`**
CEC clock source This parameter can be a value of **`RCCEEx_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 **`RCCEEx_CLK48_Clock_Source`**
- **`uint32_t RCC_PeriphCLKInitTypeDef::Sdmmc1ClockSelection`**
SDMMC1 clock source This parameter can be a value of
`RCCEEx_SDMMC1_Clock_Source`

44.2 RCCEEx 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_RCCEEx_PeriphCLKConfig()` is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register will be set to their reset values.

This section contains the following APIs:

- **`HAL_RCCEEx_PeriphCLKConfig()`**
- **`HAL_RCCEEx_GetPeriphCLKConfig()`**
- **`HAL_RCCEEx_GetPeriphCLKFreq()`**

44.2.2 `HAL_RCCEEx_PeriphCLKConfig`

Function Name	<code>HAL_StatusTypeDef HAL_RCCEEx_PeriphCLKConfig((RCC_PeriphCLKInitTypeDef * PeriphClkInit)</code>
Function Description	Initializes the RCC extended peripherals clocks according to the specified parameters in the <code>RCC_PeriphCLKInitTypeDef</code> .
Parameters	<ul style="list-style-type: none"> • PeriphClkInit: 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...).

Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • Care must be taken when HAL_RCCEEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.

44.2.3 HAL_RCCEEx_GetPeriphCLKConfig

Function Name	void HAL_RCCEEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
Function Description	Get the RCC_PeriphCLKInitTypeDef according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • PeriphClkInit: pointer to the configured RCC_PeriphCLKInitTypeDef structure
Return values	<ul style="list-style-type: none"> • None

44.2.4 HAL_RCCEEx_GetPeriphCLKFreq

Function Name	uint32_t HAL_RCCEEx_GetPeriphCLKFreq (uint32_t PeriphClk)
Function Description	Return the peripheral clock frequency for a given peripheral(SAI..)
Parameters	<ul style="list-style-type: none"> • PeriphClk: Peripheral clock identifier This parameter can be one of the following values: RCC_PERIPHCLK_SAI1: SAI1 peripheral clockRCC_PERIPHCLK_SAI2: SAI2 peripheral clock
Return values	<ul style="list-style-type: none"> • Frequency in KHz
Notes	<ul style="list-style-type: none"> • Return 0 if peripheral clock identifier not managed by this API

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

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

frequency is between Min_Data = 49 and Max_Data = 432 MHz.

__HAL_RCC_PLLI2S_CONFIG

Description:

- Macro used by the SAI HAL driver to configure the PLLI2S clock multiplication and division factors.

Parameters:

- __PLLISN: specifies the multiplication factor for PLLI2S VCO output clock. This parameter must be a number between Min_Data = 192 and Max_Data = 432.
- __PLLISQ: specifies the division factor for SAI clock. This parameter must be a number between Min_Data = 2 and Max_Data = 15.
- __PLLISR: specifies the division factor for I2S clock. This parameter must be a number between Min_Data = 2 and Max_Data = 7.
- __PLLISP: 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 PLLISN parameter correctly to ensure that the VCO output frequency is between Min_Data = 192 and Max_Data = 432 MHz.
- You have to set the PLLISR 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:

- __PLLISDivQ: specifies the PLLI2S division factor for SAI1 clock. This parameter must be a number between 1 and 32. SAI1 clock frequency = $f(\text{PLLISQ}) / \text{__PLLISDivQ}$

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: PLLSAI_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

clock.

_HAL_RCC_GET_SAI1_SOURCE

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:

- 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

_HAL_RCC_I2C2_CONFIG

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

	<p style="text-align: right;">clock</p>
<code>_HAL_RCC_GET_I2C3_SOURCE</code>	<p>Description:</p> <ul style="list-style-type: none">macro to get the I2C3 clock source. <p>Return value:</p> <ul style="list-style-type: none">The: clock source can be one of the following values:<ul style="list-style-type: none"><code>RCC_I2C3CLKSOURCE_PCLK1</code>: PCLK1 selected as I2C3 clock<code>RCC_I2C3CLKSOURCE_HSI</code>: HSI selected as I2C3 clock<code>RCC_I2C3CLKSOURCE_SYSCLK</code>: System Clock selected as I2C3 clock
<code>_HAL_RCC_I2C4_CONFIG</code>	<p>Description:</p> <ul style="list-style-type: none">Macro to configure the I2C4 clock (I2C4CLK). <p>Parameters:</p> <ul style="list-style-type: none"><code>I2C4_CLKSOURCE</code>: specifies the I2C4 clock source. This parameter can be one of the following values:<ul style="list-style-type: none"><code>RCC_I2C4CLKSOURCE_PCLK1</code>: PCLK1 selected as I2C4 clock<code>RCC_I2C4CLKSOURCE_HSI</code>: HSI selected as I2C4 clock<code>RCC_I2C4CLKSOURCE_SYSCLK</code>: System Clock selected as I2C4 clock
<code>_HAL_RCC_GET_I2C4_SOURCE</code>	<p>Description:</p> <ul style="list-style-type: none">macro to get the I2C4 clock source. <p>Return value:</p> <ul style="list-style-type: none">The: clock source can be one of the following values:<ul style="list-style-type: none"><code>RCC_I2C4CLKSOURCE_PCLK1</code>: PCLK1 selected as I2C4 clock<code>RCC_I2C4CLKSOURCE_HSI</code>: HSI selected as I2C4 clock<code>RCC_I2C4CLKSOURCE_SYSCLK</code>: System Clock selected as I2C4 clock
<code>_HAL_RCC_USART1_CONFIG</code>	<p>Description:</p> <ul style="list-style-type: none">Macro to configure the USART1 clock (USART1CLK). <p>Parameters:</p> <ul style="list-style-type: none"><code>USART1_CLKSOURCE</code>: 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_SYSCLK: System Clock selected as USART1 clock
- RCC_USART1CLKSOURCE_LSE: LSE selected as USART1 clock

__HAL_RCC_GET_USART1_SOURCE

Description:

- macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_USART1CLKSOURCE_PCLK2: PCLK2 selected as USART1 clock
 - RCC_USART1CLKSOURCE_HSI: HSI selected as USART1 clock
 - RCC_USART1CLKSOURCE_SYSCLK: System Clock selected as USART1 clock
 - RCC_USART1CLKSOURCE_LSE: LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- __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_SYSCLK: System Clock selected as USART2 clock
 - RCC_USART2CLKSOURCE_LSE: LSE selected as USART2 clock

__HAL_RCC_GET_USART2_SOURCE

Description:

- macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the

following values:

- RCC_USART2CLKSOURCE_PCLK
1: 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_PCLK
1: 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_PCLK
1: 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:

- `__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

[__HAL_RCC_GET_UART4_SOURCE](#)**Description:**

- macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
 - `RCC_UART4CLKSOURCE_PCLK1`: PCLK1 selected as UART4 clock
 - `RCC_UART4CLKSOURCE_HSI`: HSI selected as UART4 clock
 - `RCC_UART4CLKSOURCE_SYSCLK`: System Clock selected as UART4 clock
 - `RCC_UART4CLKSOURCE_LSE`: LSE selected as UART4 clock

[__HAL_RCC_UART5_CONFIG](#)**Description:**

- Macro to configure the UART5 clock (UART5CLK).

Parameters:

- `__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`

- : PCLK1 selected as UART5 clock
- RCC_UART5CLKSOURCE_HSI: HSI selected as UART5 clock
- RCC_UART5CLKSOURCE_SYSCL K: System Clock selected as UART5 clock
- RCC_UART5CLKSOURCE_LSE: LSE selected as UART5 clock

[__HAL_RCC_USART6_CONFIG](#)**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_PCLK 1: PCLK1 selected as USART6 clock
 - RCC_USART6CLKSOURCE_HSI: HSI selected as USART6 clock
 - RCC_USART6CLKSOURCE_SYSCL K: 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_PCLK 1: PCLK1 selected as USART6 clock
 - RCC_USART6CLKSOURCE_HSI: HSI selected as USART6 clock
 - RCC_USART6CLKSOURCE_SYSCL K: 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_SYSCL K: 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_SYSCL K: 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_SYSCL K: 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_SYSCL
K: 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.

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

`_HAL_RCC_CLK48_CONFIG`

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: PLSAI1 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: PLSAI1 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_CLK
48: CLK48 selected as SDMMC1
clock
- RCC_SDMMC1CLKSOURCE_SYS
CLK: 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_CRYPT_FORCE_RESET
__HAL_RCC_HASH_FORCE_RESET
__HAL_RCC_CRYPT_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
RCCEEx I2C3 Clock Source
RCC_I2C3CLKSOURCE_PCLK1
RCC_I2C3CLKSOURCE_SYSCLK
RCC_I2C3CLKSOURCE_HSI
RCCEEx I2C4 Clock Source
RCC_I2C4CLKSOURCE_PCLK1
RCC_I2C4CLKSOURCE_SYSCLK
RCC_I2C4CLKSOURCE_HSI
RCCEEx I2S Clock Source
RCC_I2SCLKSOURCE_PLLI2S
RCC_I2SCLKSOURCE_EXT
RCC Private macros to check input parameters
IS_RCC_PERIPH_CLOCK
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
```

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_RNG_CLK_ENABLE  
_HAL_RCC_USB_OTG_FS_CLK_ENABLE
```

`_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  
  
_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
```

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_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
```

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_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_CRYPT_IS_CLK_ENABLED  
_HAL_RCC_HASH_IS_CLK_ENABLED  
_HAL_RCC_CRYPT_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.

`_HAL_RCC_QSPI_IS_CLK_ENABLED`
`_HAL_RCC_FMC_IS_CLK_DISABLED`
`_HAL_RCC_QSPI_IS_CLK_DISABLED`
`_HAL_RCC_TIM2_IS_CLK_ENABLED`

`_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`

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_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
```

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_IS_CLK_ENABLED
```

```
_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_DCMI_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_DCMI_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_CRYPT_CLK_SLEEP_ENABLE  
_HAL_RCC_HASH_CLK_SLEEP_ENABLE  
_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

RCCEEx Private Defines

PLL2S_TIMEOUT_VALUE
PLLSAI_TIMEOUT_VALUE

RCCEEx SAI1 Clock Source

RCC_SAI1CLKSOURCE_PLLSAI
RCC_SAI1CLKSOURCE_PLLI2S
RCC_SAI1CLKSOURCE_PIN

RCCEEx SAI2 Clock Source

RCC_SAI2CLKSOURCE_PLLSAI
RCC_SAI2CLKSOURCE_PLLI2S
RCC_SAI2CLKSOURCE_PIN

RCCEEx SDMMC1 Clock Source

RCC_SDMMC1CLKSOURCE_CLK48
RCC_SDMMC1CLKSOURCE_SYSCLK

RCCEEx TIM Prescaler Selection

RCC_TIMPRES_DEACTIVATED
RCC_TIMPRES_ACTIVATED

RCCEEx UART4 Clock Source

RCC_UART4CLKSOURCE_PCLK1
RCC_UART4CLKSOURCE_SYSCLK
RCC_UART4CLKSOURCE_HSI
RCC_UART4CLKSOURCE_LSE

RCCEEx UART5 Clock Source

RCC_UART5CLKSOURCE_PCLK1
RCC_UART5CLKSOURCE_SYSCLK
RCC_UART5CLKSOURCE_HSI
RCC_UART5CLKSOURCE_LSE

RCCEEx UART7 Clock Source

RCC_UART7CLKSOURCE_PCLK1
RCC_UART7CLKSOURCE_SYSCLK
RCC_UART7CLKSOURCE_HSI
RCC_UART7CLKSOURCE_LSE

RCCEEx 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_MspInit\(\)`](#)
- [`HAL_RNG_MspDeInit\(\)`](#)

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	<code>HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)</code>
Function Description	Initializes the RNG peripheral and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hrng: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • HAL status

45.2.6 HAL_RNG_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)</code>
Function Description	Deinitializes the RNG peripheral.
Parameters	<ul style="list-style-type: none"> • hrng: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • HAL status

45.2.7 HAL_RNG_MspInit

Function Name	<code>void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)</code>
Function Description	Initializes the RNG MSP.
Parameters	<ul style="list-style-type: none"> • hrng: pointer to a RNG_HandleTypeDef structure that

contains the configuration information for RNG.

Return values	<ul style="list-style-type: none"> None
---------------	--

45.2.8 HAL_RNG_MspDelInit

Function Name	void HAL_RNG_MspDelInit (RNG_HandleTypeDef * hrng)
---------------	---

Function Description	DeInitializes the RNG MSP.
----------------------	----------------------------

Parameters	<ul style="list-style-type: none"> hrng: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
------------	--

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

Notes	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hrng: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
------------	--

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

45.2.11 HAL_RNG_IRQHandler

Function Name	void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)
---------------	---

Function Description	Handles RNG interrupt request.
----------------------	--------------------------------

Parameters	<ul style="list-style-type: none"> hrng: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
------------	--

Return values	<ul style="list-style-type: none"> None
---------------	--

Notes	<ul style="list-style-type: none"> 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	<code>uint32_t HAL_RNG_GetRandomNumber (RNG_HandleTypeDef * hrng)</code>
Function Description	Returns generated random number in polling mode (Obsolete) Use <code>HAL_RNG_GenerateRandomNumber()</code> API instead.
Parameters	<ul style="list-style-type: none"> • <code>hrng</code>: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • Random value

45.2.13 HAL_RNG_GetRandomNumber_IT

Function Name	<code>uint32_t HAL_RNG_GetRandomNumber_IT (RNG_HandleTypeDef * hrng)</code>
Function Description	Returns a 32-bit random number with interrupt enabled (Obsolete), Use <code>HAL_RNG_GenerateRandomNumber_IT()</code> API instead.
Parameters	<ul style="list-style-type: none"> • <code>hrng</code>: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • 32-bit random number

45.2.14 HAL_RNG_ReadLastRandomNumber

Function Name	<code>uint32_t HAL_RNG_ReadLastRandomNumber (RNG_HandleTypeDef * hrng)</code>
Function Description	Read latest generated random number.
Parameters	<ul style="list-style-type: none"> • <code>hrng</code>: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • random value

45.2.15 HAL_RNG_ReadyDataCallback

Function Name	<code>void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)</code>
Function Description	Data Ready callback in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • <code>hrng</code>: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG. • <code>random32bit</code>: generated random number.
Return values	<ul style="list-style-type: none"> • None

45.2.16 HAL_RNG_ErrorCallback

Function Name	void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)
Function Description	RNG error callbacks.
Parameters	<ul style="list-style-type: none"> • hrng: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • None

45.2.17 HAL_RNG_GetState

Function Name	HAL_RNG_StateTypeDef HAL_RNG_GetState (RNG_HandleTypeDef * hrng)
Function Description	Returns the RNG state.
Parameters	<ul style="list-style-type: none"> • hrng: pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
Return values	<ul style="list-style-type: none"> • HAL state

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

_HAL_RNG_RESET_HANDLE_STATE **Description:**

- Reset RNG handle state.

Parameters:

- **_HANDLE_**: RNG Handle

Return value:

- None

_HAL_RNG_ENABLE

Description:

- Enables the RNG peripheral.

Parameters:

- **_HANDLE_**: RNG Handle

Return value:

- None

_HAL_RNG_DISABLE**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:**

- Disables the RNG interrupts.

Parameters:

- `__HANDLE__`: RNG Handle

Return value:

- None

`__HAL_RNG_GET_IT`**Description:**

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

Parameters:

- `__HANDLE__`: RNG Handle
- `__INTERRUPT__`: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
 - `RNG_IT_DRDY`: Data ready interrupt
 - `RNG_IT_CEI`: Clock error interrupt
 - `RNG_IT_SEI`: Seed error interrupt

Return value:

- The new state of `__INTERRUPT__` (SET or RESET).

`__HAL_RNG_CLEAR_IT`**Description:**

- Clear the RNG interrupt status flags.

Parameters:

- `__HANDLE__`: RNG Handle
- `__INTERRUPT__`: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
 - `RNG_IT_CEI`: Clock error interrupt
 - `RNG_IT_SEI`: Seed error interrupt

Return value:

- None

Notes:

- `RNG_IT_DRDY` flag is read-only, reading `RNG_DR` register automatically clears `RNG_IT_DRDY`.

RNG Private Constants**`RNG_TIMEOUT_VALUE`*****RNG Private Macros*****`IS_RNG_IT`****`IS_RNG_FLAG`**

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 = 0xFFFF
- ***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_MspInit\(\)*](#)
- [*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	HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)
Function Description	Initializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

46.2.12 HAL_RTC_DeInit

Function Name	HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)
Function Description	Deinitializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- This function doesn't reset the RTC Backup Data registers.

46.2.13 HAL_RTC_MspInit

Function Name	void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)
Function Description	Initializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

46.2.14 HAL_RTC_MspDeInit

Function Name	void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)
Function Description	Deinitializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

46.2.15 HAL_RTC_SetTime

Function Name	HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Sets RTC current time.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTime: Pointer to Time structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

46.2.16 HAL_RTC_GetTime

Function Name	HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Gets RTC current time.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTime: Pointer to Time structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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.

46.2.17 HAL_RTC_SetDate

Function Name	HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function Description	Sets RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sDate: Pointer to date structure • Format: specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

46.2.18 HAL_RTC_GetDate

Function Name	HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function Description	Gets RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sDate: Pointer to Date structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

46.2.19 HAL_RTC_SetAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
---------------	--

Function Description	Sets the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. sAlarm: Pointer to Alarm structure Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> HAL status

46.2.20 HAL_RTC_SetAlarm_IT

Function Name	HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function Description	Sets the specified RTC Alarm with Interrupt.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. sAlarm: Pointer to Alarm structure Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()). The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

46.2.21 HAL_RTC_DeactivateAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)
Function Description	Deactive the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmA RTC_ALARM_B: AlarmB
Return values	<ul style="list-style-type: none"> HAL status

46.2.22 HAL_RTC_GetAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)
Function Description	Gets the RTC Alarm value and masks.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. sAlarm: Pointer to Date structure

- **Alarm:** Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmARTC_ALARM_B: AlarmB
 - **Format:** 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
- 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	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

46.2.24 HAL_RTC_AlarmAEventCallback

Function Name	void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • 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	HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)
Function Description	Returns the RTC state.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL state

46.2.28 HAL_RTC_Init

Function Name	HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)
Function Description	Initializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status

46.2.29 HAL_RTC_DeInit

Function Name	HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)
Function Description	Deinitializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> This function doesn't reset the RTC Backup Data registers.

46.2.30 HAL_RTC_MspInit

Function Name	void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)
Function Description	Initializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> None

46.2.31 HAL_RTC_MspDeInit

Function Name	void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)
---------------	--

Function Description	Deinitializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> None

46.2.32 HAL_RTC_SetTime

Function Name	HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Sets RTC current time.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. sTime: Pointer to Time structure Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> HAL status

46.2.33 HAL_RTC_GetTime

Function Name	HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Gets RTC current time.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. sTime: Pointer to Time structure Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> 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.

46.2.34 HAL_RTC_SetDate

Function Name	HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function Description	Sets RTC current date.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. sDate: Pointer to date structure Format: specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> HAL status

46.2.35 HAL_RTC_GetDate

Function Name	HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function Description	Gets RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sDate: Pointer to Date structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

46.2.36 HAL_RTC_SetAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function Description	Sets the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sAlarm: Pointer to Alarm structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

46.2.37 HAL_RTC_SetAlarm_IT

Function Name	HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function Description	Sets the specified RTC Alarm with Interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sAlarm: Pointer to Alarm structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data formatFORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).
- The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

46.2.38 HAL_RTC_DeactivateAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_DeactivateAlarm
---------------	--

(RTC_HandleTypeDef * hrtc, uint32_t Alarm)

Function Description	Deactive the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmA RTC_ALARM_B: AlarmB
Return values	<ul style="list-style-type: none"> • HAL status

46.2.39 HAL_RTC_GetAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)
Function Description	Gets the RTC Alarm value and masks.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sAlarm: Pointer to Date structure • Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: 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	<ul style="list-style-type: none"> • HAL status

46.2.40 HAL_RTC_AlarmIRQHandler

Function Name	void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	This function handles Alarm interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

46.2.41 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	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

46.2.42 HAL_RTC_AlarmAEventCallback

Function Name	void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)
---------------	--

Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

46.2.43 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	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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.

46.2.44 HAL_RTC_GetState

Function Name	HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)
Function Description	Returns the RTC state.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL state

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_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

`RTC_ALARMMASK_MINUTES`
`RTC_ALARMMASK_SECONDS`
`RTC_ALARMMASK_ALL`

RTC Alarms Definitions

`RTC_ALARM_A`
`RTC_ALARM_B`

RTC Alarm Sub Seconds Masks Definitions

<code>RTC_ALARMSUBSECONDMASK_ALL</code>	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
<code>RTC_ALARMSUBSECONDMASK_SS14_1</code>	SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
<code>RTC_ALARMSUBSECONDMASK_SS14_2</code>	SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_3</code>	SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_4</code>	SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_5</code>	SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_6</code>	SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_7</code>	SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_8</code>	SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_9</code>	SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_10</code>	SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_11</code>	SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_12</code>	SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14_13</code>	SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
<code>RTC_ALARMSUBSECONDMASK_SS14</code>	SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared
<code>RTC_ALARMSUBSECONDMASK_NONE</code>	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:

- Disable the RTC Alarm interrupt.

Parameters:

- HANDLE: specifies the RTC handle.
- INTERRUPT: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
 - RTC_IT_ALRA: Alarm A interrupt
 - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

[__HAL_RTC_ALARM_GET_IT](#)

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- HANDLE: specifies the RTC handle.
- INTERRUPT: specifies the RTC Alarm interrupt to check. This parameter can be:
 - RTC_IT_ALRA: Alarm A interrupt
 - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

[__HAL_RTC_ALARM_GET_FLAG](#)

- Get the selected RTC Alarm's flag status.

Parameters:

- HANDLE: specifies the RTC handle.
- FLAG: specifies the RTC Alarm Flag to check. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRBF

- RTC_FLAG_ALRAW
- F
- RTC_FLAG_ALRBW
- F

Return value:

- None

[__HAL_RTC_ALARM_CLEAR_FLAG](#)

Description:

- Clear the RTC Alarm's pending flags.

Parameters:

- [__HANDLE__](#): specifies the RTC handle.
- [__FLAG__](#): specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRBF

Return value:

- None

[__HAL_RTC_ALARM_GET_IT_SOURCE](#)

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- [__HANDLE__](#): specifies the RTC handle.
- [__INTERRUPT__](#): specifies the RTC Alarm interrupt sources to check. This parameter can be:
 - RTC_IT_ALRA: Alarm A interrupt
 - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

[__HAL_RTC_ALARM_EXTI_ENABLE_IT](#)

Description:

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

`_HAL_RTC_ALARM_EXTI_DISABLE_IT`

Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

`_HAL_RTC_ALARM_EXTI_ENABLE_EVENT`

Description:

- Enable event on the RTC Alarm associated Exti line.

Return value:

- None.

`_HAL_RTC_ALARM_EXTI_DISABLE_EVENT`

Description:

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None.

`_HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE`

Description:

- Enable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

`_HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE`

Description:

- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

`_HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE`

Description:

- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

`_HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE`

Description:

- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

<code>__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE</code>	Description: <ul style="list-style-type: none">Enable rising & falling edge trigger on the RTC Alarm associated Exti line. Return value: <ul style="list-style-type: none">None.
<code>__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE</code>	Description: <ul style="list-style-type: none">Disable rising & falling edge trigger on the RTC Alarm associated Exti line. Return value: <ul style="list-style-type: none">None.
<code>__HAL_RTC_ALARM_EXTI_GET_FLAG</code>	Description: <ul style="list-style-type: none">Check whether the RTC Alarm associated Exti line interrupt flag is set or not. Return value: <ul style="list-style-type: none">Line: Status.
<code>__HAL_RTC_ALARM_EXTI_CLEAR_FLAG</code>	Description: <ul style="list-style-type: none">Clear the RTC Alarm associated Exti line flag. Return value: <ul style="list-style-type: none">None.
<code>__HAL_RTC_ALARM_EXTI_GENERATE_SWIT</code>	Description: <ul style="list-style-type: none">Generate a Software interrupt on RTC Alarm associated Exti line. Return value: <ul style="list-style-type: none">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_APRIIL
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_PUSH_PULL

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 RTCEEx 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
[**RTCEEx_Tamper_Pins_Definitions**](#)
- ***uint32_t RTC_TamperTypeDef::Interrupt***
Specifies the Tamper Interrupt. This parameter can be a value of
[**RTCEEx_Tamper_Interrupt_Definitions**](#)
- ***uint32_t RTC_TamperTypeDef::Trigger***
Specifies the Tamper Trigger. This parameter can be a value of
[**RTCEEx_Tamper_Trigger_Definitions**](#)
- ***uint32_t RTC_TamperTypeDef::NoErase***
Specifies the Tamper no erase mode. This parameter can be a value of
[**RTCEEx_Tamper_EraseBackUp_Definitions**](#)
- ***uint32_t RTC_TamperTypeDef::MaskFlag***
Specifies the Tamper Flag masking. This parameter can be a value of
[**RTCEEx_Tamper_MaskFlag_Definitions**](#)
- ***uint32_t RTC_TamperTypeDef::Filter***
Specifies the RTC Filter Tamper. This parameter can be a value of
[**RTCEEx_Tamper_Filter_Definitions**](#)
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
Specifies the sampling frequency. This parameter can be a value of
[**RTCEEx_Tamper_Sampling_Frequencies_Definitions**](#)
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
Specifies the Precharge Duration . This parameter can be a value of
[**RTCEEx_Tamper_Pin_Precharge_Duration_Definitions**](#)
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
Specifies the Tamper PullUp . This parameter can be a value of
[**RTCEEx_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_TAMPCCR 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_RTCEEx_SetTimeStamp\(\)`](#)
- [`HAL_RTCEEx_SetTimeStamp_IT\(\)`](#)
- [`HAL_RTCEEx_DeactivateTimeStamp\(\)`](#)
- [`HAL_RTCEEx_SetInternalTimeStamp\(\)`](#)
- [`HAL_RTCEEx_DeactivateInternalTimeStamp\(\)`](#)
- [`HAL_RTCEEx_GetTimeStamp\(\)`](#)
- [`HAL_RTCEEx_SetTamper\(\)`](#)
- [`HAL_RTCEEx_SetTamper_IT\(\)`](#)
- [`HAL_RTCEEx_DeactivateTamper\(\)`](#)
- [`HAL_RTCEEx_TamperTimeStampIRQHandler\(\)`](#)
- [`HAL_RTCEEx_TimeStampEventCallback\(\)`](#)
- [`HAL_RTCEEx_Tamper1EventCallback\(\)`](#)
- [`HAL_RTCEEx_Tamper2EventCallback\(\)`](#)
- [`HAL_RTCEEx_Tamper3EventCallback\(\)`](#)
- [`HAL_RTCEEx_PollForTimeStampEvent\(\)`](#)
- [`HAL_RTCEEx_PollForTamper1Event\(\)`](#)
- [`HAL_RTCEEx_PollForTamper2Event\(\)`](#)
- [`HAL_RTCEEx_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_RTCEEx_SetWakeUpTimer\(\)`](#)
- [`HAL_RTCEEx_SetWakeUpTimer_IT\(\)`](#)
- [`HAL_RTCEEx_DeactivateWakeUpTimer\(\)`](#)
- [`HAL_RTCEEx_GetWakeUpTimer\(\)`](#)
- [`HAL_RTCEEx_WakeUpTimerIRQHandler\(\)`](#)
- [`HAL_RTCEEx_WakeUpTimerEventCallback\(\)`](#)
- [`HAL_RTCEEx_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 Alram 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	<code>HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp((RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)</code>
Function Description	Sets TimeStamp.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • TimeStampEdge: Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin. • RTC_TimeStampPin: specifies the RTC TimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_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.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This API must be called before enabling the TimeStamp feature.

47.2.7 HAL_RTCEx_SetTimeStamp_IT



Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)
Function Description	Sets TimeStamp with Interrupt.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. TimeStampEdge: Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin. RTC_TimeStampPin: Specifies the RTC TimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_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.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> This API must be called before enabling the TimeStamp feature.

47.2.8 HAL_RTCEx_DeactivateTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates TimeStamp.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status

47.2.9 HAL_RTCEx_SetInternalTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)
Function Description	Sets Internal TimeStamp.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status

Notes

- This API must be called before enabling the internal TimeStamp feature.

47.2.10 HAL_RTCEx_DeactivateInternalTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates internal TimeStamp.

Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.11 HAL_RTCEx_GetTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)
Function Description	Gets the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTimeStamp: Pointer to Time structure • sTimeStampDate: Pointer to Date structure • Format: specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

47.2.12 HAL_RTCEx_SetTamper

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Sets Tamper.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTamper: Pointer to Tamper Structure.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • By calling this API we disable the tamper interrupt for all tampers.

47.2.13 HAL_RTCEx_SetTamper_IT

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTamper: Pointer to RTC Tamper.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • By calling this API we force the tamper interrupt for all tampers.

47.2.14 HAL_RTCEx_DeactivateTamper

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper
---------------	---

(RTC_HandleTypeDef * hrtc, uint32_t Tamper)

Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Tamper: Selected tamper pin. This parameter can be RTC_Tamper_1 and/or RTC_TAMPER_2.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.15 HAL_RTCEx_TamperTimeStampIRQHandler

Function Name	void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	This function handles TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

47.2.16 HAL_RTCEx_TimeStampEventCallback

Function Name	void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

47.2.17 HAL_RTCEx_Tamper1EventCallback

Function Name	void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

47.2.18 HAL_RTCEx_Tamper2EventCallback

Function Name	void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

47.2.19 HAL_RTCEx_Tamper3EventCallback

Function Name	void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 3 callback.
Parameters	<ul style="list-style-type: none"> hrtc: RTC handle
Return values	<ul style="list-style-type: none"> None
47.2.20 HAL_RTCEx_PollForTimeStampEvent	
Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles TimeStamp polling request.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
47.2.21 HAL_RTCEx_PollForTamper1Event	
Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper1 Polling.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
47.2.22 HAL_RTCEx_PollForTamper2Event	
Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper2 Polling.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
47.2.23 HAL_RTCEx_PollForTamper3Event	
Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper3 Polling.
Parameters	<ul style="list-style-type: none"> hrtc: RTC handle Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
47.2.24 HAL_RTCEx_SetWakeUpTimer	

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Sets wake up timer.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. WakeUpCounter: Wake up counter WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none"> HAL status

47.2.25 HAL_RTCEx_SetWakeUpTimer_IT

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Sets wake up timer with interrupt.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. WakeUpCounter: Wake up counter WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none"> HAL status

47.2.26 HAL_RTCEx_DeactivateWakeUpTimer

Function Name	uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates wake up timer counter.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status

47.2.27 HAL_RTCEx_GetWakeUpTimer

Function Name	uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)
Function Description	Gets wake up timer counter.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> Counter value

47.2.28 HAL_RTCEx_WakeUpTimerIRQHandler

Function Name	void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	This function handles Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values	<ul style="list-style-type: none"> None
47.2.29 HAL_RTCEx_WakeUpTimerEventCallback	
Function Name	void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Read value

47.2.33 HAL_RTCEx_SetSmoothCalib

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)</code>
Function Description	Sets the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> • 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*11 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

47.2.34 HAL_RTCEx_SetSynchroShift

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)</code>
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • ShiftAdd1S: Select to add or not 1 second to the time calendar. This parameter can be one of the following values : RTC_SHIFTADD1S_SET: Add one second to the clock calendar.RTC_SHIFTADD1S_RESET: No effect. • ShiftSubFS: Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When REFCKON is set, firmware must not write to Shift control register.

47.2.35 HAL_RTCEx_SetCalibrationOutPut

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • CalibOutput: Select the Calibration output Selection . This parameter can be one of the following values: RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.36 HAL_RTCEx_DeactivateCalibrationOutPut

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.37 HAL_RTCEx_SetRefClock

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Enables the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.38 HAL_RTCEx_DeactivateRefClock

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Disable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.39 HAL_RTCEx_EnableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)
---------------	--

Function Description	Enables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

47.2.40 HAL_RTCEx_DisableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow(RTC_HandleTypeDef * hrtc)
Function Description	Disables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

47.2.41 HAL_RTCEx_AlarmBEventCallback

Function Name	void HAL_RTCEx_AlarmBEventCallback(RTC_HandleTypeDef * hrtc)
Function Description	Alarm B callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

47.2.42 HAL_RTCEx_PollForAlarmBEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent(RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles AlarmB Polling request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

47.2.43 HAL_RTCEx_SetTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp(RTC_HandleTypeDef * hrtc, uint32_tTimeStampEdge, uint32_t RTC_TimeStampPin)
Function Description	Sets TimeStamp.
Parameters	<ul style="list-style-type: none"> • 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 RTCTimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_PC13: PC13 is selected as RTCTimeStamp Pin.RTC_TIMESTAMPPIN_PI8: PI8 is selected as RTCTimeStamp Pin.RTC_TIMESTAMPPIN_PC1: PC1 is selected as RTCTimeStamp Pin.

Return values

- HAL status

Notes

- This API must be called before enabling theTimeStamp feature.

47.2.44 HAL_RTCEx_SetTimeStamp_IT

Function Name

**HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT
(RTC_HandleTypeDef * hrtc, uint32_tTimeStampEdge,
uint32_t RTC_TimeStampPin)**

Function Description

SetsTimeStamp with Interrupt.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **TimeStampEdge:** Specifies the pin edge on which theTimeStamp 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 RTCTimeStamp Pin. This parameter can be one of the following values: RTC_TIMESTAMPPIN_PC13: PC13 is selected as RTCTimeStamp Pin.RTC_TIMESTAMPPIN_PI8: PI8 is selected as RTCTimeStamp Pin.RTC_TIMESTAMPPIN_PC1: PC1 is selected as RTCTimeStamp Pin.

Return values

- HAL status

Notes

- This API must be called before enabling theTimeStamp feature.

47.2.45 HAL_RTCEx_DeactivateTimeStamp

Function Name

**HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp
(RTC_HandleTypeDef * hrtc)**

Function Description

DeactivatesTimeStamp.

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	HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)
Function Description	Sets Internal TimeStamp.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This API must be called before enabling the internal TimeStamp feature.

47.2.47 HAL_RTCEx_DeactivateInternalTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates internal TimeStamp.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.48 HAL_RTCEx_GetTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)
Function Description	Gets the RTC TimeStamp value.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTimeStamp: Pointer to Time structure • sTimeStampDate: Pointer to Date structure • Format: specifies the format of the entered parameters. This parameter can be one of the following values: FORMAT_BIN: Binary data format FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

47.2.49 HAL_RTCEx_SetTamper

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Sets Tamper.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTamper: Pointer to Tamper Structure.
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- By calling this API we disable the tamper interrupt for all tampers.

47.2.50 HAL_RTCEx_SetTamper_IT

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTamper: Pointer to RTC Tamper.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • By calling this API we force the tamper interrupt for all tampers.

47.2.51 HAL_RTCEx_DeactivateTamper

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)
Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Tamper: Selected tamper pin. This parameter can be RTC_Tamper_1 and/or RTC_TAMPER_2.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.52 HAL_RTCEx_TamperTimeStampIRQHandler

Function Name	void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	This function handles TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

47.2.53 HAL_RTCEx_Tamper1EventCallback

Function Name	void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 1 callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

47.2.54 HAL_RTCEx_Tamper2EventCallback

Function Name	void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 2 callback.

Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> None

47.2.55 HAL_RTCEx_Tamper3EventCallback

Function Name	void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 3 callback.
Parameters	<ul style="list-style-type: none"> hrtc: RTC handle
Return values	<ul style="list-style-type: none"> None

47.2.56 HAL_RTCEx_TimeStampEventCallback

Function Name	void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> None

47.2.57 HAL_RTCEx_PollForTimeStampEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles TimeStamp polling request.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

47.2.58 HAL_RTCEx_PollForTamper1Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper1 Polling.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

47.2.59 HAL_RTCEx_PollForTamper2Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper2 Polling.

Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

47.2.60 HAL_RTCEx_PollForTamper3Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper3 Polling.
Parameters	<ul style="list-style-type: none"> hrtc: RTC handle Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

47.2.61 HAL_RTCEx_SetWakeUpTimer

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Sets wake up timer.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. WakeUpCounter: Wake up counter WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none"> HAL status

47.2.62 HAL_RTCEx_SetWakeUpTimer_IT

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Sets wake up timer with interrupt.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. WakeUpCounter: Wake up counter WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none"> HAL status

47.2.63 HAL_RTCEx_DeactivateWakeUpTimer

Function Name	uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates wake up timer counter.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status

47.2.64 HAL_RTCEx_GetWakeUpTimer

Function Name	<code>uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)</code>
Function Description	Gets wake up timer counter.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none">• Counter value

47.2.65 HAL_RTCEx_WakeUpTimerIRQHandler

Function Name	<code>void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)</code>
Function Description	This function handles Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none">• None

47.2.66 HAL_RTCEx_WakeUpTimerEventCallback

Function Name	<code>void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)</code>
Function Description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none">• None

47.2.67 HAL_RTCEx_PollForWakeUpTimerEvent

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)</code>
Function Description	This function handles Wake Up Timer Polling.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.• Timeout: Timeout duration
Return values	<ul style="list-style-type: none">• HAL status

47.2.68 HAL_RTCEx_BKUPWrite

Function Name	<code>void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)</code>
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none">• 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

47.2.69 HAL_RTCEx_BKUPRead

Function Name	<code>uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)</code>
Function Description	Reads data from the specified RTC Backup data Register.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Read value

47.2.70 HAL_RTCEx_SetSmoothCalib

Function Name	<code>HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)</code>
Function Description	Sets the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • SmoothCalibPeriod: Select the Smooth Calibration Period. This parameter can be 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^{11} 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	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • 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	HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. ShiftAdd1S: Select to add or not 1 second to the time calendar. This parameter can be one of the following values : RTC_SHIFTADD1S_SET: Add one second to the clock calendar.RTC_SHIFTADD1S_RESET: No effect. ShiftSubFS: Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When REFCKON is set, firmware must not write to Shift control register.

47.2.72 HAL_RTCEx_SetCalibrationOutPut

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. CalibOutput: Select the Calibration output Selection . This parameter can be one of the following values: RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.
Return values	<ul style="list-style-type: none"> HAL status

47.2.73 HAL_RTCEx_DeactivateCalibrationOutPut

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status

47.2.74 HAL_RTCEx_SetRefClock

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Enables the RTC reference clock detection.

Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

47.2.75 HAL_RTCEx_DeactivateRefClock

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Disable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	

47.2.76 HAL_RTCEx_EnableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Enables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

47.2.77 HAL_RTCEx_DisableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Disables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

47.2.78 HAL_RTCEx_AlarmBEventCallback

Function Name	void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Alarm B callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

47.2.79 HAL_RTCEx_PollForAlarmBEvent

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles AlarmB Polling request.
Parameters	<ul style="list-style-type: none">hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.Timeout: Timeout duration
Return values	<ul style="list-style-type: none">HAL status

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

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- `_HANDLE_`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- `_HANDLE_`: specifies the RTC

handle.

- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer 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:

- 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

[__HAL_RTC_WAKEUPTIMER_GET_FLAG](#)**Description:**

- Get the selected RTC WakeUpTimer's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
 - RTC_FLAG_WU
TF
 - RTC_FLAG_WU
TWF

Return value:

- None

[__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG](#)**Description:**

- Clear the RTC Wake

Up timer's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
 - RTC_FLAG_WU TF

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.

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

Tamper interrupt to check. This parameter can be:

- RTC_IT_TAMP:
All tampers
interrupts
- RTC_IT_TAMP1:
Tamper1
interrupt
- RTC_IT_TAMP2:
Tamper2
interrupt
- RTC_IT_TAMP3:
Tamper3
interrupt

Return value:

- None

[__HAL_RTC_TAMPER_GET_IT_SOURCE](#)

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- [__HANDLE__](#): specifies the RTC handle.
- [__INTERRUPT__](#): specifies the RTC Tamper interrupt source to check. This parameter can be:
 - RTC_IT_TAMP:
All tampers
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

`_HAL_RTC_TAMPER_CLEAR_FLAG`**Description:**

- Clear the RTC Tamper's pending flags.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag sources to 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

`_HAL_RTC_TIMESTAMP_ENABLE`**Description:**

- Enable the RTC TimeStamp peripheral.

Parameters:

- `__HANDLE__`:

specifies the RTC handle.

Return value:

- None

`__HAL_RTC_TIMESTAMP_DISABLE`

Description:

- Disable the RTC TimeStamp peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_TIMESTAMP_ENABLE_IT`

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
 - `RTC_IT_TS`: TimeStamp interrupt

Return value:

- None

`__HAL_RTC_TIMESTAMP_DISABLE_IT`

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:

- RTC_IT_TS:
TimeStamp
interrupt

Return value:

- None

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

[__HAL_RTC_TIMESTAMP_GET_IT_SOURCE](#)**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

[__HAL_RTC_TIMESTAMP_GET_FLAG](#)**Description:**

- Get the selected RTC TimeStamp's flag

status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
 - RTC_FLAG_TSF
 - RTC_FLAG_TSO_VF

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_TSO_VF

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.

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_ITS_F

Return value:

- None

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_ITS_F

Return value:

- None

Description:

- Enable the RTC calibration output.

Parameters:

- __HANDLE__:

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_WAKEUPTIMER_EXTI_ENABLE_IT`

- 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

WakeUp Timer
associated Exti line.

Return value:

- None.

Description:

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

Description:

- Clear the RTC

WakeUp Timer
associated Exti line
flag.

Return value:

- None.

Description:

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

Description:

- Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

Description:

- Enable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

Description:

- Disable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

Description:

- Enable falling edge

trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_FALLING_EDGE`

Description:

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE`

Description:

- Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE`

Description:

- Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_EDGE`

Description:

- Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_FALLING_EDGE`

Description:

- Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`_HAL_RTC_TAMPER_TIMESTAMP_EXTI_GET_FLAG`**Description:**

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

`_HAL_RTC_TAMPER_TIMESTAMP_EXTI_CLEAR_FLAG`**Description:**

- Clear the RTC Tamper and Timestamp associated Exti line flag.

Return value:

- None.

`_HAL_RTC_TAMPER_TIMESTAMP_EXTI_GENERATE_SWIT`**Description:**

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

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_TIMESTAMPONRTAMPERT_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 $2^{\text{exp}20}$ RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_16SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else $2^{\text{exp}19}$ RTCCLK seconds
RTC_SMOOTHCALIB_PERIOD_8SEC	If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else $2^{\text{exp}18}$ RTCCLK seconds

RTCEx Smooth calib Plus pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET	The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8
RTC_SMOOTHCALIB_PLUSPULSES_RESET	The number of RTCCLK pulses substituted during a 32-second window = 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.
<i>RTCEx Tamper Interrupt Definitions</i>	
RTC_TAMPER1_INTERRUPT	
RTC_TAMPER2_INTERRUPT	
RTC_TAMPER3_INTERRUPT	
RTC_ALL_TAMPER_INTERRUPT	
<i>RTCEx Tamper MaskFlag Definitions</i>	
RTC_TAMPERMASK_FLAG_DISABLE	
RTC_TAMPERMASK_FLAG_ENABLE	
<i>RTCEx Tamper Pins Definitions</i>	
RTC_TAMPER_1	
RTC_TAMPER_2	
RTC_TAMPER_3	
<i>RTCEx Tamper Pin Precharge Duration Definitions</i>	
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
<i>RTCEx Tamper Pull UP Definitions</i>	
RTC_TAMPER_PULLUP_ENABLE	TimeStamp on Tamper Detection event saved
RTC_TAMPER_PULLUP_DISABLE	TimeStamp on Tamper Detection event is not saved
<i>RTCEx Tamper Sampling Frequencies Definitions</i>	
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768	Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384	Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192	Each of the tamper inputs are sampled with a frequency =

	RTCCLK / 8192
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096	Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048	Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024	Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512	Each of the tamper inputs are sampled with a frequency = RTCCLK / 512
RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256	Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

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 Time Stamp 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_TRIState_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_FrameInitTypeDef

Data Fields

- ***uint32_t FrameLength***
- ***uint32_t ActiveFrameLength***
- ***uint32_t FSDefinition***
- ***uint32_t FSPolarity***
- ***uint32_t FOffset***

Field Documentation

- ***uint32_t SAI_FrameInitTypeDef::FrameLength***
Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.
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_FrameInitTypeDef::ActiveFrameLength***
Specifies the Frame synchronization active level length. This Parameter specifies the

- length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame.
This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t SAI_FrameInitTypeDef::FSDefinition***
Specifies the Frame synchronization definition. This parameter can be a value of [**SAI_Block_FS_Definition**](#)
- ***uint32_t SAI_FrameInitTypeDef::FSPolarity***
Specifies the Frame synchronization Polarity. This parameter can be a value of [**SAI_Block_FS_Polarity**](#)
- ***uint32_t SAI_FrameInitTypeDef::FSOffset***
Specifies the Frame synchronization Offset. This parameter can be a value of [**SAI_Block_FS_Offset**](#)

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_FrameInitTypeDef FrameInit***
- ***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_Block_TypeDef* __SAI_HandleTypeDefDef::Instance***
SAI Blockx registers base address
- ***SAI_InitTypeDef __SAI_HandleTypeDefDef::Init***
SAI communication parameters
- ***SAI_FrameInitTypeDef __SAI_HandleTypeDefDef::FrameInit***
SAI Frame configuration parameters
- ***SAI_SlotInitTypeDef __SAI_HandleTypeDefDef::SlotInit***
SAI Slot configuration parameters
- ***uint8_t* __SAI_HandleTypeDefDef::pBuffPtr***
Pointer to SAI transfer Buffer
- ***uint16_t __SAI_HandleTypeDefDef::XferSize***
SAI transfer size
- ***uint16_t __SAI_HandleTypeDefDef::XferCount***
SAI transfer counter
- ***DMA_HandleTypeDef* __SAI_HandleTypeDefDef::hdmatx***
SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SAI_HandleTypeDefDef::hdmarx***
SAI Rx DMA handle parameters
- ***SAIcallback __SAI_HandleTypeDefDef::mutecallback***
SAI mute callback
- ***void(* __SAI_HandleTypeDefDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDefDef *hsai)***
- ***HAL_LockTypeDef __SAI_HandleTypeDefDef::Lock***
SAI locking object
- ***_IO HAL_SAI_StateTypeDef __SAI_HandleTypeDefDef::State***
SAI communication state
- ***_IO uint32_t __SAI_HandleTypeDefDef::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 <= (SLOT size - Data size)
- Data size <= SLOT size
- Number of SLOT x SLOT size = Frame length
- The number of slots should be even when SAI_FS_CHANNEL_IDENTIFICATION is selected.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SAI_Transmit()

- Receive an amount of data in blocking mode using HAL_SAI_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_SAI_Transmit_IT()
- At transmission end of transfer HAL_SAI_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_SAI_Receive_IT()
- At reception end of transfer HAL_SAI_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback
- In case of transfer Error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_SAI_Transmit_DMA()
- At transmission end of transfer HAL_SAI_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_SAI_Receive_DMA()
- At reception end of transfer HAL_SAI_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback
- In case of transfer Error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback
- Pause the DMA Transfer using HAL_SAI_DMAPause()
- Resume the DMA Transfer using HAL_SAI_DMAResume()
- Stop the DMA Transfer using HAL_SAI_DMAPause()

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	<code>HAL_StatusTypeDef HAL_SAI_InitProtocol(SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)</code>
Function Description	Initializes the structure Framelnit, 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"> • hsai: : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • protocol: : one of the supported protocol SAI Supported protocol • datasize: : one of the supported datasize SAI protocol data size the configuration information for SAI module. • nbslot: : Number of slot.
Return values	<ul style="list-style-type: none"> • HAL status

48.2.6 HAL_SAI_Init

Function Name	<code>HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)</code>
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"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • HAL status

48.2.7 HAL_SAI_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_SAI_DeInit (SAI_HandleTypeDef * hsai)</code>
Function Description	Deinitializes the SAI peripheral.

Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • HAL status

48.2.8 HAL_SAI_MspInit

Function Name	void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)
Function Description	SAI MSP Init.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • None

48.2.9 HAL_SAI_MspDelInit

Function Name	void HAL_SAI_MspDelInit (SAI_HandleTypeDef * hsai)
Function Description	SAI MSP DelInit.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • None

48.2.10 HAL_SAI_Transmit

Function Name	HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

48.2.11 HAL_SAI_Receive

Function Name	HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • pData: Pointer to data buffer • Size: Amount of data to be received • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

48.2.12 HAL_SAI_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SAI_Transmit_IT
---------------	--

(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"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

48.2.13 HAL_SAI_Receive_IT

Function Name	HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)
Function Description	Receives an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • pData: Pointer to data buffer • Size: Amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status

48.2.14 HAL_SAI_DMAPause

Function Name	HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)
Function Description	Pauses the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • HAL status

48.2.15 HAL_SAI_DMAResume

Function Name	HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • HAL status

48.2.16 HAL_SAI_DMAStop

Function Name	HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)
Function Description	Stops the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • HAL status

48.2.17 HAL_SAI_Abort

Function Name	<code>HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)</code>
Function Description	Abort the current transfer and disable the SAI.
Parameters	<ul style="list-style-type: none"> • hsai: : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • HAL status

48.2.18 HAL_SAI_Transmit_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)</code>
Function Description	Transmits an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

48.2.19 HAL_SAI_Receive_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)</code>
Function Description	Receives an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • pData: Pointer to data buffer • Size: Amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status

48.2.20 HAL_SAI_EnableTxMuteMode

Function Name	<code>HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)</code>
Function Description	Enable the tx mute mode.
Parameters	<ul style="list-style-type: none"> • hsai: : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module. • val: : value sent during the mute SAI Block Mute Value
Return values	<ul style="list-style-type: none"> • HAL status

48.2.21 HAL_SAI_DisableTxMuteMode

Function Name	<code>HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)</code>
Function Description	Disable the tx mute mode.
Parameters	<ul style="list-style-type: none"> • hsai: : pointer to a SAI_HandleTypeDef structure that

contains the configuration information for SAI module.

Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hsai: : pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • HAL status

48.2.25 HAL_SAI_TxCpltCallback

Function Name	void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> • None

48.2.26 HAL_SAI_TxHalfCpltCallback

Function Name	void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)
Function Description	Tx Transfer Half completed callbacks.
Parameters	<ul style="list-style-type: none"> • hsai: pointer to a SAI_HandleTypeDef structure that contains

the configuration information for SAI module.

Return values	<ul style="list-style-type: none"> None
---------------	--

48.2.27 HAL_SAI_RxCpltCallback

Function Name	void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> None

48.2.28 HAL_SAI_RxHalfCpltCallback

Function Name	void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)
Function Description	Rx Transfer half completed callbacks.
Parameters	<ul style="list-style-type: none"> hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> None

48.2.29 HAL_SAI_ErrorCallback

Function Name	void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)
Function Description	SAI error callbacks.
Parameters	<ul style="list-style-type: none"> hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> hsai: pointer to a SAI_HandleTypeDef structure that contains the configuration information for the specified SAI Block.
Return values	<ul style="list-style-type: none"> 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_FIFO_THRESHOLD_EMPTY
SAI_FIFO_THRESHOLD_1QF
SAI_FIFO_THRESHOLD_HF
SAI_FIFO_THRESHOLD_3QF
SAI_FIFO_THRESHOLD_FULL

SAI Block Flags Definition

SAI_FLAG_OVRUDR
SAI_FLAG_MUTEDET
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_MUTEDET
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

<code>HAL_SAI_ERROR_NONE</code>	No error
---------------------------------	----------

<code>HAL_SAI_ERROR_OVR</code>	Overrun Error
--------------------------------	---------------

<code>HAL_SAI_ERROR_UDR</code>	Underrun error
--------------------------------	----------------

<code>HAL_SAI_ERROR_AFSDET</code>	Anticipated Frame synchronisation detection
-----------------------------------	---

<code>HAL_SAI_ERROR_LFSDET</code>	Late Frame synchronisation detection
-----------------------------------	--------------------------------------

<code>HAL_SAI_ERROR_CNREADY</code>	codec not ready
------------------------------------	-----------------

<code>HAL_SAI_ERROR_WCKCFG</code>	Wrong clock configuration
-----------------------------------	---------------------------

<code>HAL_SAI_ERROR_TIMEOUT</code>	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

__HAL_SAI_DISABLE_IT
__HAL_SAI_GET_IT_SOURCE

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).

__HAL_SAI_GET_FLAG

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:

- 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).

[__HAL_SAI_CLEAR_FLAG](#)**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_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 TRIS State 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_HandleTypeDefDef * hsai)**

Function Description Configure SAI Block synchronization mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDefDef 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_HandleTypeDefDef * hsai)**

Function Description Get SAI Input Clock based on SAI source clock selection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDefDef 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_HandleTypeDefDef

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_HandleTypeDefDef::Instance***
Register base address
- ***FMC_SDRAM_InitTypeDef SDRAM_HandleTypeDefDef::Init***
SDRAM device configuration parameters
- ***__IO HAL_SDRAM_StateTypeDef SDRAM_HandleTypeDefDef::State***
SDRAM access state
- ***HAL_LockTypeDef SDRAM_HandleTypeDefDef::Lock***
SDRAM locking object
- ***DMA_HandleTypeDef* SDRAM_HandleTypeDefDef::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_HandleTypeDefDef 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_DelInit()`
- `HAL_SDRAM_MsplInit()`
- `HAL_SDRAM_MspDelInit()`
- `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	HAL_StatusTypeDef HAL_SDRAM_Init (SDRAM_HandleTypeDef * hsdran, FMC_SDRAM_TimingTypeDef * Timing)
Function Description	Performs the SDRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> • hsdran: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module. • Timing: Pointer to SDRAM control timing structure
Return values	<ul style="list-style-type: none"> • HAL status

50.2.7 HAL_SDRAM_DelInit

Function Name	HAL_StatusTypeDef HAL_SDRAM_DelInit (SDRAM_HandleTypeDef * hsdran)
Function Description	Perform the SDRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> • hsdran: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
Return values	<ul style="list-style-type: none"> • HAL status

50.2.8 HAL_SDRAM_MspInit

Function Name	void HAL_SDRAM_MspInit (SDRAM_HandleTypeDef * hsdran)
Function Description	SDRAM MSP Init.
Parameters	<ul style="list-style-type: none"> • hsdran: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
Return values	<ul style="list-style-type: none"> • None

50.2.9 HAL_SDRAM_MspDelInit

Function Name	void HAL_SDRAM_MspDelInit (SDRAM_HandleTypeDef * hsdran)
Function Description	SDRAM MSP DelInit.
Parameters	<ul style="list-style-type: none"> • hsdran: pointer to a SDRAM_HandleTypeDef structure that

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_RefreshErrorHandler

Function Name **void HAL_SDRAM_RefreshErrorHandler (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_XferErrorHandler

Function Name **void HAL_SDRAM_DMA_XferErrorHandler (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
- HAL status

Return values

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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status
50.2.22 HAL_SDRAM_WriteProtection_Enable	
Function Name	HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Enable (SDRAM_HandleTypeDef * hsdr)
Function Description	Enables dynamically SDRAM write protection.
Parameters	<ul style="list-style-type: none"> • hsdr: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
Return values	<ul style="list-style-type: none"> • HAL status
50.2.23 HAL_SDRAM_WriteProtection_Disable	
Function Name	HAL_StatusTypeDef HAL_SDRAM_WriteProtection_Disable (SDRAM_HandleTypeDef * hsdr)
Function Description	Disables dynamically SDRAM write protection.
Parameters	<ul style="list-style-type: none"> • hsdr: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
Return values	<ul style="list-style-type: none"> • HAL status
50.2.24 HAL_SDRAM_SendCommand	
Function Name	HAL_StatusTypeDef HAL_SDRAM_SendCommand (SDRAM_HandleTypeDef * hsdr, FMC_SDRAM_CommandTypeDef * Command, uint32_t Timeout)
Function Description	Sends Command to the SDRAM bank.
Parameters	<ul style="list-style-type: none"> • hsdr: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module. • Command: SDRAM command structure • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
50.2.25 HAL_SDRAM_ProgramRefreshRate	
Function Name	HAL_StatusTypeDef HAL_SDRAM_ProgramRefreshRate (SDRAM_HandleTypeDef * hsdr, uint32_t RefreshRate)
Function Description	Programs the SDRAM Memory Refresh rate.
Parameters	<ul style="list-style-type: none"> • hsdr: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module. • RefreshRate: The SDRAM refresh rate value
Return values	<ul style="list-style-type: none"> • HAL status
50.2.26 HAL_SDRAM_SetAutoRefreshNumber	
Function Name	HAL_StatusTypeDef HAL_SDRAM_SetAutoRefreshNumber (SDRAM_HandleTypeDef * hsdr, uint32_t AutoRefreshNumber)

Function Description	Sets the Number of consecutive SDRAM Memory auto Refresh commands.
Parameters	<ul style="list-style-type: none"> • hsdram: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module. • AutoRefreshNumber: The SDRAM auto Refresh number
Return values	<ul style="list-style-type: none"> • HAL status

50.2.27 HAL_SDRAM_GetModeStatus

Function Name	<code>uint32_t HAL_SDRAM_GetModeStatus (SDRAM_HandleTypeDef * hsdram)</code>
Function Description	Returns the SDRAM memory current mode.
Parameters	<ul style="list-style-type: none"> • hsdram: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
Return values	<ul style="list-style-type: none"> • The SDRAM memory mode.

50.2.28 HAL_SDRAM_GetState

Function Name	<code>HAL_SDRAM_StateTypeDef HAL_SDRAM_GetState (SDRAM_HandleTypeDef * hsdram)</code>
Function Description	Returns the SDRAM state.
Parameters	<ul style="list-style-type: none"> • hsdram: pointer to a SDRAM_HandleTypeDef structure that contains the configuration information for SDRAM module.
Return values	<ul style="list-style-type: none"> • HAL state

50.3 SDRAM Firmware driver defines

50.3.1 SDRAM

SDRAM Exported Macros

<code>_HAL_SDRAM_RESET_HANDLE_STATE</code>	Description:
	<ul style="list-style-type: none"> • Reset SDRAM handle state.
	Parameters:
	<ul style="list-style-type: none"> • <code>_HANDLE_</code>: specifies the SDRAM handle.
	Return value:
	<ul style="list-style-type: none"> • None

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 ManDefIECC`
- `__IO uint8_t WrSpeedFact`
- `__IO uint8_t MaxWrBlockLen`
- `__IO uint8_t WriteBlockPaPartial`
- `__IO uint8_t Reserved3`
- `__IO uint8_t ContentProtectAppli`
- `__IO uint8_t FileFormatGrouop`
- `__IO uint8_t CopyFlag`
- `__IO uint8_t PermWrProtect`
- `__IO uint8_t TempWrProtect`
- `__IO uint8_t FileFormat`
- `__IO uint8_t ECC`
- `__IO uint8_t CSD_CRC`
- `__IO uint8_t Reserved4`

Field Documentation

- `__IO uint8_t HAL_SD_CSDTypedef::CSDStruct`
CSD structure
- `__IO uint8_t HAL_SD_CSDTypedef::SysSpecVersion`
System specification version
- `__IO uint8_t HAL_SD_CSDTypedef::Reserved1`
Reserved

- **`_IO uint8_t HAL_SD_CSDTypeDef::TAAC`**
Data read access time 1
- **`_IO uint8_t HAL_SD_CSDTypeDef::NSAC`**
Data read access time 2 in CLK cycles
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxBusClkFrec`**
Max. bus clock frequency
- **`_IO uint16_t HAL_SD_CSDTypeDef::CardComdClasses`**
Card command classes
- **`_IO uint8_t HAL_SD_CSDTypeDef::RdBlockLen`**
Max. read data block length
- **`_IO uint8_t HAL_SD_CSDTypeDef::PartBlockRead`**
Partial blocks for read allowed
- **`_IO uint8_t HAL_SD_CSDTypeDef::WrBlockMisalign`**
Write block misalignment
- **`_IO uint8_t HAL_SD_CSDTypeDef::RdBlockMisalign`**
Read block misalignment
- **`_IO uint8_t HAL_SD_CSDTypeDef::DSRImpl`**
DSR implemented
- **`_IO uint8_t HAL_SD_CSDTypeDef::Reserved2`**
Reserved
- **`_IO uint32_t HAL_SD_CSDTypeDef::DeviceSize`**
Device Size
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxRdCurrentVDDMin`**
Max. read current @ VDD min
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxRdCurrentVDDMax`**
Max. read current @ VDD max
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxWrCurrentVDDMin`**
Max. write current @ VDD min
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxWrCurrentVDDMax`**
Max. write current @ VDD max
- **`_IO uint8_t HAL_SD_CSDTypeDef::DeviceSizeMul`**
Device size multiplier
- **`_IO uint8_t HAL_SD_CSDTypeDef::EraseGrSize`**
Erase group size
- **`_IO uint8_t HAL_SD_CSDTypeDef::EraseGrMul`**
Erase group size multiplier
- **`_IO uint8_t HAL_SD_CSDTypeDef::WrProtectGrSize`**
Write protect group size
- **`_IO uint8_t HAL_SD_CSDTypeDef::WrProtectGrEnable`**
Write protect group enable
- **`_IO uint8_t HAL_SD_CSDTypeDef::ManDefIECC`**
Manufacturer default ECC
- **`_IO uint8_t HAL_SD_CSDTypeDef::WrSpeedFact`**
Write speed factor
- **`_IO uint8_t HAL_SD_CSDTypeDef::MaxWrBlockLen`**
Max. write data block length
- **`_IO uint8_t HAL_SD_CSDTypeDef::WriteBlockPaPartial`**
Partial blocks for write allowed
- **`_IO uint8_t HAL_SD_CSDTypeDef::Reserved3`**
Reserved
- **`_IO uint8_t HAL_SD_CSDTypeDef::ContentProtectAppli`**
Content protection application
- **`_IO uint8_t HAL_SD_CSDTypeDef::FileFormatGrouop`**
File format group

- `_IO uint8_t HAL_SD_CSDTypedef::CopyFlag`
Copy flag (OTP)
- `_IO uint8_t HAL_SD_CSDTypedef::PermWrProtect`
Permanent write protection
- `_IO uint8_t HAL_SD_CSDTypedef::TempWrProtect`
Temporary write protection
- `_IO uint8_t HAL_SD_CSDTypedef::FileFormat`
File format
- `_IO uint8_t HAL_SD_CSDTypedef::ECC`
ECC code
- `_IO uint8_t HAL_SD_CSDTypedef::CSD_CRC`
CSD CRC
- `_IO uint8_t HAL_SD_CSDTypedef::Reserved4`
Always 1

51.1.3 HAL_SD_CIDTypedef

Data Fields

- `_IO uint8_t ManufacturerID`
- `_IO uint16_t OEM_AppId`
- `_IO uint32_t ProdName1`
- `_IO uint8_t ProdName2`
- `_IO uint8_t ProdRev`
- `_IO uint32_t ProdSN`
- `_IO uint8_t Reserved1`
- `_IO uint16_t ManufactDate`
- `_IO uint8_t CID_CRC`
- `_IO uint8_t Reserved2`

Field Documentation

- `_IO uint8_t HAL_SD_CIDTypedef::ManufacturerID`
Manufacturer ID
- `_IO uint16_t HAL_SD_CIDTypedef::OEM_AppId`
OEM/Application ID
- `_IO uint32_t HAL_SD_CIDTypedef::ProdName1`
Product Name part1
- `_IO uint8_t HAL_SD_CIDTypedef::ProdName2`
Product Name part2
- `_IO uint8_t HAL_SD_CIDTypedef::ProdRev`
Product Revision
- `_IO uint32_t HAL_SD_CIDTypedef::ProdSN`
Product Serial Number
- `_IO uint8_t HAL_SD_CIDTypedef::Reserved1`
Reserved1
- `_IO uint16_t HAL_SD_CIDTypedef::ManufactDate`
Manufacturing Date
- `_IO uint8_t HAL_SD_CIDTypedef::CID_CRC`
CID CRC

- `_IO uint8_t HAL_SD_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 = SDMMCCCLK / (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_DelInit\(\)*](#)
- [*HAL_SD_MspInit\(\)*](#)
- [*HAL_SD_MspDelInit\(\)*](#)

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	<code>HAL_SD_ErrorTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * SDCardInfo)</code>
Function Description	Initializes the SD card according to the specified parameters in the <code>SD_HandleTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • SDCardInfo: <code>HAL_SD_CardInfoTypeDef</code> structure for SD card information
Return values	<ul style="list-style-type: none"> • HAL SD error state

51.2.7 HAL_SD_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)</code>
Function Description	De-Initializes the SD card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • HAL status

51.2.8 HAL_SD_MspInit

Function Name	<code>void HAL_SD_MspInit (SD_HandleTypeDef * hsd)</code>
Function Description	Initializes the SD MSP.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

51.2.9 HAL_SD_MspDeInit

Function Name	<code>void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)</code>
Function Description	De-Initialize SD MSP.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

51.2.10 HAL_SD_ReadBlocks

Function Name	<code>HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</code>
Function Description	Reads block(s) from a specified address in a card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pReadBuffer: pointer to the buffer that will contain the received data • ReadAddr: Address from where data is to be read • BlockSize: SD card Data block size • NumberOfBlocks: Number of SD blocks to read
Return values	<ul style="list-style-type: none"> • SD Card error state
Notes	<ul style="list-style-type: none"> • BlockSize must be 512 bytes.

51.2.11 HAL_SD_WriteBlocks

Function Name	<code>HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</code>
Function Description	Allows to write block(s) to a specified address in a card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pWriteBuffer: pointer to the buffer that will contain the data to transmit • WriteAddr: Address from where data is to be written • BlockSize: SD card Data block size • NumberOfBlocks: Number of SD blocks to write
Return values	<ul style="list-style-type: none"> • SD Card error state
Notes	<ul style="list-style-type: none"> • BlockSize must be 512 bytes.

51.2.12 HAL_SD_ReadBlocks_DMA

Function Name	<code>HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)</code>
Function Description	Reads block(s) from a specified address in a card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pReadBuffer: Pointer to the buffer that will contain the received data • ReadAddr: Address from where data is to be read • BlockSize: SD card Data block size • NumberOfBlocks: Number of blocks to read.
Return values	<ul style="list-style-type: none"> • SD Card error state
Notes	<ul style="list-style-type: none"> • This API should be followed by the function <code>HAL_SD_CheckReadOperation()</code> to check the completion of the read process • BlockSize must be 512 bytes.

51.2.13 HAL_SD_WriteBlocks_DMA

Function Name	HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Writes block(s) to a specified address in a card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pWriteBuffer: pointer to the buffer that will contain the data to transmit • WriteAddr: Address from where data is to be read • BlockSize: the SD card Data block size • NumberOfBlocks: Number of blocks to write
Return values	<ul style="list-style-type: none"> • SD Card error state
Notes	<ul style="list-style-type: none"> • This API should be followed by the function HAL_SD_CheckWriteOperation() to check the completion of the write process (by SD current status polling). • BlockSize must be 512 bytes.

51.2.14 HAL_SD_CheckReadOperation

Function Name	HAL_SD_ErrorTypeDef HAL_SD_CheckReadOperation (SD_HandleTypeDef * hsd, uint32_t Timeout)
Function Description	This function waits until the SD DMA data read transfer is finished.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • SD Card error state

51.2.15 HAL_SD_CheckWriteOperation

Function Name	HAL_SD_ErrorTypeDef HAL_SD_CheckWriteOperation (SD_HandleTypeDef * hsd, uint32_t Timeout)
Function Description	This function waits until the SD DMA data write transfer is finished.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • SD Card error state

51.2.16 HAL_SD_Erase

Function Name	HAL_SD_ErrorTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint64_t startaddr, uint64_t endaddr)
Function Description	Erases the specified memory area of the given SD card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • startaddr: Start byte address • endaddr: End byte address
Return values	<ul style="list-style-type: none"> • SD Card error state

51.2.17 HAL_SD_IRQHandler

Function Name	void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)
Function Description	This function handles SD card interrupt request.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

51.2.18 HAL_SD_XferCpltCallback

Function Name	void HAL_SD_XferCpltCallback (SD_HandleTypeDef * hsd)
Function Description	SD end of transfer callback.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

51.2.19 HAL_SD_XferErrorCallback

Function Name	void HAL_SD_XferErrorCallback (SD_HandleTypeDef * hsd)
Function Description	SD Transfer Error callback.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

51.2.20 HAL_SD_DMA_RxCpltCallback

Function Name	void HAL_SD_DMA_RxCpltCallback (DMA_HandleTypeDef * hdma)
Function Description	SD Transfer complete Rx callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> • None

51.2.21 HAL_SD_DMA_RxErrorCallback

Function Name	void HAL_SD_DMA_RxErrorCallback (DMA_HandleTypeDef * hdma)
Function Description	SD DMA transfer complete Rx error callback.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> • None

51.2.22 HAL_SD_DMA_TxCpltCallback

Function Name	void HAL_SD_DMA_TxCpltCallback (DMA_HandleTypeDef * hdma)
Function Description	SD Transfer complete Tx callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that

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	HAL_SD_ErrorTypeDef HAL_SD_HighSpeed (SD_HandleTypeDef * hsd)
Function Description	Switches the SD card to High Speed mode.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • SD Card error state
Notes	<ul style="list-style-type: none"> • This operation should be followed by the configuration of PLL to have SDMMCCK clock between 67 and 75 MHz

51.2.28 HAL_SD_SendSDStatus

Function Name	HAL_SD_ErrorTypeDef HAL_SD_SendSDStatus (SD_HandleTypeDef * hsd, uint32_t * pSDstatus)
Function Description	Returns the current SD card's status.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pSDstatus: Pointer to the buffer that will contain the SD card status SD Status register)
Return values	<ul style="list-style-type: none"> • SD Card error state

51.2.29 HAL_SD_GetStatus

Function Name	HAL_SD_TransferStateTypeDef HAL_SD_GetStatus (SD_HandleTypeDef * hsd)
Function Description	Gets the current sd card data status.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • Data Transfer state

51.2.30 HAL_SD_GetCardStatus

Function Name	HAL_SD_ErrorTypeDef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pCardStatus)
Function Description	Gets the SD card status.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pCardStatus: Pointer to the HAL_SD_CardStatusTypeDef structure that will contain the SD card status information
Return values	<ul style="list-style-type: none"> • SD Card error state

51.3 SD Firmware driver defines

51.3.1 SD

SD Exported Constants

SD_CMD_GO_IDLE_STATE	Resets the SD memory card.
----------------------	----------------------------

SD_CMD_SEND_OP_COND	Sends host capacity support information and activates the card's initialization process.
SD_CMD_ALL_SEND_CID	Asks any card connected to the host to send the CID numbers on the CMD line.
SD_CMD_SET_REL_ADDR	Asks the card to publish a new relative address (RCA).
SD_CMD_SET_DSR	Programs the DSR of all cards.
SD_CMD_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

range to be erased.

SD_CMD_ERASE_GRP_START	Sets the address of the first write block to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE	Reserved for SD security applications.
SD_CMD_FAST_IO	SD card doesn't support it (Reserved).
SD_CMD_GO_IRQ_STATE	SD card doesn't support it (Reserved).
SD_CMD_LOCK_UNLOCK	Sets/resets the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
SD_CMD_APP_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
SD_CMD_GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general purpose/application specific commands.
SD_CMD_NO_CMD	
SD_CMD_APP_SD_SET_BUSWIDTH	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_MMICARD	

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_CMDSENT`: 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_TXFIFOF: Transmit FIFO full interrupt
- SDMMC_IT_RXFIFOF: 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_CMDSENT: 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_TXFIFOF: Transmit FIFO full interrupt
- SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
- SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDMMC_IT_RXFIFOE: Receive FIFO empty interrupt
- SDMMC_IT_TXDABL: Data available in transmit FIFO interrupt
- SDMMC_IT_RXDABL: 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_CMDSENT: 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_TXFIFOF: Transmit FIFO full
- SDMMC_FLAG_RXFIFOF: 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_CMDSENT: 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_CMDSENT: 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

- 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_TXFIFOF: Transmit FIFO full interrupt
- SDMMC_IT_RXFIFOF: 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).

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 configures 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_DelInit\(\)`](#)
- [`HAL_SMARTCARD_MspInit\(\)`](#)
- [`HAL_SMARTCARD_MspDelInit\(\)`](#)

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	HAL_StatusTypeDef HAL_SMARTCARD_Init(SMARTCARD_HandleTypeDef * hsc)
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"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • HAL status

52.2.6 HAL_SMARTCARD_DelInit

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_DelInit(SMARTCARD_HandleTypeDef * hsc)
Function Description	Delinitializes the SMARTCARD peripheral.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • HAL status

52.2.7 HAL_SMARTCARD_MspInit

Function Name	void HAL_SMARTCARD_MspInit(SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD MSP Init.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • None

52.2.8 HAL_SMARTCARD_MspDelInit

Function Name	void HAL_SMARTCARD_MspDelInit(SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD MSP DelInit.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • None

52.2.9 HAL_SMARTCARD_Transmit

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

52.2.10 HAL_SMARTCARD_Receive

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be received • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

52.2.11 HAL_SMARTCARD_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

52.2.12 HAL_SMARTCARD_Receive_IT

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status

52.2.13 HAL_SMARTCARD_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t
---------------	---

Size)

Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

52.2.14 HAL_SMARTCARD_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA(SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle • pData: pointer to data buffer • Size: amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

52.2.15 HAL_SMARTCARD_IRQHandler

Function Name	void HAL_SMARTCARD_IRQHandler(SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD interrupt requests handling.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • None

52.2.16 HAL_SMARTCARD_TxCpltCallback

Function Name	void HAL_SMARTCARD_TxCpltCallback(SMARTCARD_HandleTypeDef * hsc)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • None

52.2.17 HAL_SMARTCARD_RxCpltCallback

Function Name	void HAL_SMARTCARD_RxCpltCallback(SMARTCARD_HandleTypeDef * hsc)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • None

52.2.18 HAL_SMARTCARD_ErrorCallback

Function Name	void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD error callbacks.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hsc: : pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD.
Return values	<ul style="list-style-type: none"> • 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:

- None

_HAL_SMARTCARD_GET_FLAG

Description:

- Checks whether the specified Smartcard flag is set or not.

Parameters:

- _HANDLE_: specifies the SMARTCARD Handle. The Handle Instance which can be USART1 or USART2.
- _FLAG_: specifies the flag to check. This parameter can be one of the following values:
 - SMARTCARD_FLAG_RXACK: 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

_HAL_SMARTCARD_DISABLE_IT

- 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

_HAL_SMARTCARD_GET_IT**Description:**

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

Parameters:

- __HANDLE__: specifies the SMARTCARD

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
CE

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

- 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

- 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

Return value:

- None

_HAL_SMARTCARD_ENABLE**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

_HAL_SMARTCARD_DISABLE**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

_HAL_SMARTCARD_DMA_REQUEST_ENABLE**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_DISABLE***SMARTCARD_GTPR_GT_LSBPOS*****SMARTCARD_GTPR_GT_LSB_POS*****SMARTCARD Interruption Mask*****SMARTCARD_IT_MASK*****SMARTCARD Interrupt 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	<code>void HAL_SMARTCARDEX_BlockLength_Config(SMARTCARD_HandleTypeDef * hsc, uint8_t BlockLength)</code>
Function Description	Update on the fly the SMARTCARD block length in RTOR register.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle • BlockLength: SMARTCARD block length (8-bit long at most)
Return values	<ul style="list-style-type: none"> • None

53.1.4 `HAL_SMARTCARDEX_TimeOut_Config`

Function Name	<code>void HAL_SMARTCARDEX_TimeOut_Config(SMARTCARD_HandleTypeDef * hsc, uint32_t TimeOutValue)</code>
Function Description	Update on the fly the receiver timeout value in RTOR register.
Parameters	<ul style="list-style-type: none"> • hsc: SMARTCARD handle • TimeOutValue: receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0xFFFFFFFF.

Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">hsc: SMARTCARD handle
Return values	<ul style="list-style-type: none">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_HandleTypeDefDef

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 (IDLE 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	HAL_StatusTypeDef HAL_SPDIFRX_Init (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"> • hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> • HAL status

54.2.6 HAL_SPDIFRX_DeInit

Function Name	HAL_StatusTypeDef HAL_SPDIFRX_DeInit (SPDIFRX_HandleTypeDef * hspdif)
Function Description	DeInitializes the SPDIFRX peripheral.
Parameters	<ul style="list-style-type: none"> • hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> • HAL status

54.2.7 HAL_SPDIFRX_MspInit

Function Name	void HAL_SPDIFRX_MspInit (SPDIFRX_HandleTypeDef * hspdif)
Function Description	SPDIFRX MSP Init.

Parameters	<ul style="list-style-type: none"> hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> None

54.2.8 HAL_SPDIFRX_MspDelInit

Function Name	void HAL_SPDIFRX_MspDelInit (SPDIFRX_HandleTypeDef * hspdif)
Function Description	SPDIFRX MSP DelInit.
Parameters	<ul style="list-style-type: none"> hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> None

54.2.9 HAL_SPDIFRX_SetDataFormat

Function Name	HAL_StatusTypeDef HAL_SPDIFRX_SetDataFormat (SPDIFRX_HandleTypeDef * hspdif, SPDIFRX_SetDataFormatTypeDef sDataFormat)
Function Description	Sets the SPDIFRX data format according to the specified parameters in the SPDIFRX_InitTypeDef.
Parameters	<ul style="list-style-type: none"> hspdif: SPDIFRX handle sDataFormat: SPDIFRX data format
Return values	<ul style="list-style-type: none"> HAL status

54.2.10 HAL_SPDIFRX_ReceiveDataFlow

Function Name	HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives an amount of data (Data Flow) in blocking mode.
Parameters	<ul style="list-style-type: none"> hspdif: pointer to SPDIFRX_HandleTypeDef structure that contains the configuration information for SPDIFRX module. pData: Pointer to data buffer Size: Amount of data to be received Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

54.2.11 HAL_SPDIFRX_ReceiveControlFlow

Function Name	HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow (SPDIFRX_HandleTypeDef * hspdif, uint32_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives an amount of data (Control Flow) in blocking mode.
Parameters	<ul style="list-style-type: none"> hspdif: pointer to a SPDIFRX_HandleTypeDef structure that contains the configuration information for SPDIFRX module. pData: Pointer to data buffer Size: Amount of data to be received Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

54.2.12 HAL_SPDIFRX_ReceiveDataFlow_IT

Function Name	HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_IT (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"> • hspdif: SPDIFRX handle • pData: a 32-bit pointer to the Receive data buffer. • Size: number of data sample to be received .
Return values	<ul style="list-style-type: none"> • HAL status

54.2.13 HAL_SPDIFRX_ReceiveControlFlow_IT

Function Name	HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow_IT (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"> • hspdif: SPDIFRX handle • pData: a 32-bit pointer to the Receive data buffer. • Size: number of data sample (Control Flow) to be received :
Return values	<ul style="list-style-type: none"> • HAL status

54.2.14 HAL_SPDIFRX_ReceiveDataFlow_DMA

Function Name	HAL_StatusTypeDef HAL_SPDIFRX_ReceiveDataFlow_DMA (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"> • hspdif: SPDIFRX handle • pData: a 32-bit pointer to the Receive data buffer. • Size: number of data sample to be received :
Return values	<ul style="list-style-type: none"> • HAL status

54.2.15 HAL_SPDIFRX_ReceiveControlFlow_DMA

Function Name	HAL_StatusTypeDef HAL_SPDIFRX_ReceiveControlFlow_DMA (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"> • hspdif: SPDIFRX handle • pData: a 32-bit pointer to the Receive data buffer. • Size: number of data (Control Flow) sample to be received :
Return values	<ul style="list-style-type: none"> • HAL status

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	<ul style="list-style-type: none"> • hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • hspdif: SPDIFRX handle

Return values	<ul style="list-style-type: none"> None
---------------	--

54.2.22 HAL_SPDIFRX_ErrorCallback

Function Name	<code>void HAL_SPDIFRX_ErrorCallback (SPDIFRX_HandleTypeDef * hspdif)</code>
Function Description	SPDIFRX error callbacks.
Parameters	<ul style="list-style-type: none"> hspdif: SPDIFRX handle
Return values	<ul style="list-style-type: none"> None

54.2.23 HAL_SPDIFRX_GetState

Function Name	<code>HAL_SPDIFRX_StateTypeDef HAL_SPDIFRX_GetState (SPDIFRX_HandleTypeDef * hspdif)</code>
Function Description	Return the SPDIFRX state.
Parameters	<ul style="list-style-type: none"> hspdif: : SPDIFRX handle
Return values	<ul style="list-style-type: none"> HAL state

54.2.24 HAL_SPDIFRX_GetError

Function Name	<code>uint32_t HAL_SPDIFRX_GetError (SPDIFRX_HandleTypeDef * hspdif)</code>
Function Description	Return the SPDIFRX error code.
Parameters	<ul style="list-style-type: none"> hspdif: : SPDIFRX handle
Return values	<ul style="list-style-type: none"> 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

<code>_HAL_SPDIFRX_RESET_HANDLE_STATE</code>	Description:
	<ul style="list-style-type: none"> • Reset SPDIFRX handle state. Parameters: <ul style="list-style-type: none"> • <code>_HANDLE_</code>: SPDIFRX handle. Return value: <ul style="list-style-type: none"> • None
<code>_HAL_SPDIFRX_IDLE</code>	Description:
	<ul style="list-style-type: none"> • Disable the specified SPDIFRX peripheral (IDLE State). Parameters: <ul style="list-style-type: none"> • <code>_HANDLE_</code>: specifies the SPDIFRX Handle. Return value: <ul style="list-style-type: none"> • None
<code>_HAL_SPDIFRX_SYNC</code>	Description:
	<ul style="list-style-type: none"> • Enable the specified SPDIFRX peripheral (SYNC State). Parameters: <ul style="list-style-type: none"> • <code>_HANDLE_</code>: specifies the SPDIFRX Handle. Return value: <ul style="list-style-type: none"> • None
<code>_HAL_SPDIFRX_RCV</code>	Description:
	<ul style="list-style-type: none"> • Enable the specified SPDIFRX peripheral (RCV State). Parameters: <ul style="list-style-type: none"> • <code>_HANDLE_</code>: specifies the SPDIFRX Handle. Return value: <ul style="list-style-type: none"> • None
<code>_HAL_SPDIFRX_ENABLE_IT</code>	Description:
	<ul style="list-style-type: none"> • Enable or disable the specified SPDIFRX interrupts.

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

__HAL_SPDIFRX_DISABLE_IT
__HAL_SPDIFRX_GET_IT_SOURCE

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).

__HAL_SPDIFRX_GET_FLAG

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:

- 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).

[_HAL_SPDIFRX_CLEAR_IT](#)**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

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_SYNC DIE

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_STEREO MODE_DISABLE

SPDIFRX_STEREO MODE_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::TIMode***
Specifies if the TI mode is enabled or not . This parameter can be a value of ***SPI_TIMode***
- ***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
 - TIMode
 - CRC Calculation
 - CRC Polynomial if CRC enabled
 - CRC Length, used only with Data8 and Data16
 - FIFO reception threshold
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx 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_DMAPause\(\)**](#)

- [`HAL_SPI_DMAResume\(\)`](#)
- [`HAL_SPI_DMAStop\(\)`](#)
- [`HAL_SPI_IRQHandler\(\)`](#)
- [`HAL_SPI_TxCpltCallback\(\)`](#)
- [`HAL_SPI_RxCpltCallback\(\)`](#)
- [`HAL_SPI_TxRxCpltCallback\(\)`](#)
- [`HAL_SPI_TxHalfCpltCallback\(\)`](#)
- [`HAL_SPI_RxHalfCpltCallback\(\)`](#)
- [`HAL_SPI_TxRxHalfCpltCallback\(\)`](#)
- [`HAL_SPI_ErrorCallback\(\)`](#)

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	<code>HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)</code>
Function Description	Initializes the SPI according to the specified parameters in the <code>SPI_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

55.2.6 HAL_SPI_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)</code>
Function Description	DeInitializes the SPI peripheral.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

55.2.7 HAL_SPI_MspInit

Function Name	<code>void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)</code>
Function Description	SPI MSP Init.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

55.2.8 HAL_SPI_MspDeInit

Function Name	void HAL_SPI_MspDelInit (SPI_HandleTypeDef * hspi)
Function Description	SPI MSP DelInit.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> None

55.2.9 HAL_SPI_Transmit

Function Name	HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. pData: pointer to data buffer Size: amount of data to be sent Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

55.2.10 HAL_SPI_Receive

Function Name	HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> 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 received Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

55.2.11 HAL_SPI_TransmitReceive

Function Name	HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Transmit and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> 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 Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

55.2.12 HAL_SPI_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SPI_Transmit_IT
---------------	--

(SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function Description	Transmit an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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"> • hspi: SPI handle • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

55.2.17 HAL_SPI_TransmitReceive_DMA

Function Name	HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Transmit and Receive an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pTxData: pointer to transmission data buffer • pRxData: pointer to reception data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the CRC feature is enabled the pRxData Length must be Size + 1

55.2.18 HAL_SPI_DMAPause

Function Name	HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

55.2.19 HAL_SPI_DMAResume

Function Name	HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

55.2.20 HAL_SPI_DMAStop

Function Name	HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)
Function Description	Stops the DMA Transfer.

Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none">• HAL status

55.2.21 HAL_SPI_IRQHandler

Function Name	void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)
Function Description	This function handles SPI interrupt request.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none">• None

55.2.22 HAL_SPI_TxCpltCallback

Function Name	void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

55.2.23 HAL_SPI_RxCpltCallback

Function Name	void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

55.2.24 HAL_SPI_TxRxCpltCallback

Function Name	void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx and Rx Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

55.2.25 HAL_SPI_TxHalfCpltCallback

Function Name	void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

55.2.26 HAL_SPI_RxHalfCpltCallback

Function Name	void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Rx Half Transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

55.2.27 HAL_SPI_TxRxHalfCpltCallback

Function Name	void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx and Rx Half Transfer callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

55.2.28 HAL_SPI_ErrorCallback

Function Name	void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)
Function Description	SPI error callback.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

55.2.29 HAL_SPI_GetState

Function Name	HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)
Function Description	Return the SPI state.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • SPI state

55.2.30 HAL_SPI_GetError

Function Name	uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)
Function Description	Return the SPI error code.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • SPI error code in bitmap format

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

<code>_HAL_SPI_RESET_HANDLE_STATE</code>	Description: <ul style="list-style-type: none"> Reset SPI handle state. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: SPI handle. Return value: <ul style="list-style-type: none"> None
<code>_HAL_SPI_ENABLE_IT</code>	Description: <ul style="list-style-type: none"> Enables or disables the specified SPI interrupts. Parameters: <ul style="list-style-type: none"> <code>_HANDLE_</code>: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral. <code>_INTERRUPT_</code>: specifies the interrupt source to enable or disable. This parameter can be one of the following values: <ul style="list-style-type: none"> <code>SPI_IT_TXE</code>: Tx buffer empty interrupt enable <code>SPI_IT_RXNE</code>: RX buffer not empty interrupt enable <code>SPI_IT_ERR</code>: Error interrupt enable Return value: <ul style="list-style-type: none"> None
<code>_HAL_SPI_DISABLE_IT</code>	
<code>_HAL_SPI_GET_IT_SOURCE</code>	Description: <ul style="list-style-type: none"> Checks if the specified SPI interrupt source is enabled or disabled.

Parameters:

- HANDLE: : specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- INTERRUPT: : specifies the SPI interrupt source to check. This parameter can be one of the following values:
 - SPI_IT_TXE: Tx buffer empty interrupt enable
 - SPI_IT_RXNE: RX buffer not empty interrupt enable
 - SPI_IT_ERR: Error interrupt enable

Return value:

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

_HAL_SPI_GET_FLAG**Description:**

- Checks whether the specified SPI flag is set or not.

Parameters:

- HANDLE: : specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- FLAG: : specifies the flag to check. This parameter can be one of the following values:
 - SPI_FLAG_RXNE: Receive buffer not empty flag
 - SPI_FLAG_TXE: Transmit buffer empty flag
 - SPI_FLAG_CRCERR: CRC error flag
 - SPI_FLAG_MODF: Mode fault flag
 - SPI_FLAG_OVR: Overrun flag
 - SPI_FLAG_BSY: Busy flag
 - SPI_FLAG_FRE: Frame format error flag
 - SPI_FLAG_FTLVL: SPI fifo transmission level
 - SPI_FLAG_FRLVL: SPI fifo reception level

Return value:

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

_HAL_SPI_CLEAR_CRCERRFLAG**Description:**

- Clears the SPI CRCERR pending flag.

Parameters:

- HANDLE: : specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

[__HAL_SPI_CLEAR_MODFLAG](#)

Description:

- Clears the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

[__HAL_SPI_CLEAR_OVRFLAG](#)

Description:

- Clears the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

[__HAL_SPI_CLEAR_FREFLAG](#)

Description:

- Clears the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

[__HAL_SPI_ENABLE](#)

Description:

- Enables the SPI.

Parameters:

- __HANDLE__: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

[__HAL_SPI_DISABLE](#)

Description:

- Disables the SPI.

Parameters:

- __HANDLE__: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

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

SPI_1LINE_RX

Description:

- Sets the SPI receive-only mode.

Parameters:

- `_HANDLE_`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

SPI_RESET_CRC

Description:

- Resets the CRC calculation of the SPI.

Parameters:

- `_HANDLE_`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

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 hsramp; and:
 - Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
 - Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
 - Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two 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	<code>HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)</code>
Function Description	Performs the SRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • Timing: Pointer to SRAM control timing structure • ExtTiming: Pointer to SRAM extended mode timing structure
Return values	<ul style="list-style-type: none"> • HAL status

56.2.7 HAL_SRAM_DelInit

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_DelInit (SRAM_HandleTypeDef * hsram)</code>
Function Description	Performs the SRAM device De-initialization sequence.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • HAL status

56.2.8 HAL_SRAM_MspInit

Function Name	<code>void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)</code>
Function Description	SRAM MSP Init.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • None

56.2.9 HAL_SRAM_MspDelInit

Function Name	<code>void HAL_SRAM_MspDelInit (SRAM_HandleTypeDef * hsram)</code>
Function Description	SRAM MSP DelInit.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • None

56.2.10 HAL_SRAM_DMA_XferCpltCallback

Function Name	<code>void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)</code>
Function Description	DMA transfer complete callback.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • None

56.2.11 HAL_SRAM_DMA_XferErrorCallback

Function Name	<code>void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)</code>
Function Description	DMA transfer complete error callback.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • None

56.2.12 HAL_SRAM_Read_8b

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)</code>
Function Description	Reads 8-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

56.2.13 HAL_SRAM_Write_8b

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)</code>
Function Description	Writes 8-bit buffer to SRAM memory.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

56.2.14 HAL_SRAM_Read_16b

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t</code>
---------------	---

*** pDstBuffer, uint32_t BufferSize)**

Function Description	Reads 16-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

56.2.18 HAL_SRAM_Read_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)</code>
Function Description	Reads a Words data from the SRAM memory using DMA transfer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

56.2.19 HAL_SRAM_Write_DMA

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)</code>
Function Description	Writes a Words data buffer to SRAM memory using DMA transfer.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

56.2.20 HAL_SRAM_DMA_XferCpltCallback

Function Name	<code>void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)</code>
Function Description	DMA transfer complete callback.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • None

56.2.21 HAL_SRAM_DMA_XferErrorCallback

Function Name	<code>void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)</code>
Function Description	DMA transfer complete error callback.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • None

56.2.22 HAL_SRAM_WriteOperation_Enable

Function Name	<code>HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)</code>
---------------	--

Function Description	Enables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • HAL status

56.2.23 HAL_SRAM_WriteOperation_Disable

Function Name	HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable(SRAM_HandleTypeDef * hsram)
Function Description	Disables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • HAL status

56.2.24 HAL_SRAM_GetState

Function Name	HAL_SRAM_StateTypeDef HAL_SRAM_GetState(SRAM_HandleTypeDef * hsram)
Function Description	Returns the SRAM controller state.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • HAL state

56.3 SRAM Firmware driver defines

56.3.1 SRAM

SRAM Exported Macros

`_HAL_SRAM_RESET_HANDLE_STATE` **Description:**

- Reset SRAM handle state.

Parameters:

- `_HANDLE_`: SRAM handle

Return value:

- None

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 OCIdleState*
- *uint32_t OCNPolarity*
- *uint32_t OCNPolarity*
- *uint32_t OCFastMode*
- *uint32_t OCIdleState*
- *uint32_t OCIdleState*

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
Specifies the TIM mode. This parameter can be a value of
[**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 IC_Polarity***
- ***uint32_t IC_Selection***
- ***uint32_t IC_Filter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
Specifies the TIM mode. This parameter can be a value of
[**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::OC_Polarity***
Specifies the output polarity. This parameter can be a value of [**TIM_Output_Compare_Polarity**](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::OC_N_Polarity***
Specifies the complementary output polarity. This parameter can be a value of [**TIM_Output_Compare_N_Polarity**](#)
Note:This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::OC_Idle_State***
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [**TIM_Output_Compare_Idle_State**](#)
Note:This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCN_Idle_State***
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [**TIM_Output_Compare_N_Idle_State**](#)
Note:This parameter is valid only for TIM1 and TIM8.
- ***uint32_t TIM_OnePulse_InitTypeDef::IC_Polarity***
Specifies the active edge of the input signal. This parameter can be a value of [**TIM_Input_Capture_Polarity**](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::IC_Selection***
Specifies the input. This parameter can be a value of [**TIM_Input_Capture_Selection**](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::IC_Filter***
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

57.1.4 TIM_IC_InitTypeDef

Data Fields

- ***uint32_t IC_Polarity***
- ***uint32_t IC_Selection***
- ***uint32_t IC_Prescaler***
- ***uint32_t IC_Filter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::IC_Polarity***
Specifies the active edge of the input signal. This parameter can be a value of [**TIM_Input_Capture_Polarity**](#)
- ***uint32_t TIM_IC_InitTypeDef::IC_Selection***
Specifies the input. This parameter can be a value of [**TIM_Input_Capture_Selection**](#)
- ***uint32_t TIM_IC_InitTypeDef::IC_Prescaler***
Specifies the Input Capture Prescaler. This parameter can be a value of [**TIM_Input_Capture_Prescaler**](#)
- ***uint32_t TIM_IC_InitTypeDef::IC_Filter***
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

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	<code>HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the <code>TIM_HandleTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • <code>htim</code>: pointer to a <code>TIM_HandleTypeDef</code> structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

57.2.14 HAL_TIM_Base_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM Base peripheral.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

57.2.15 HAL_TIM_Base_MspInit

Function Name	void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Base MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.16 HAL_TIM_Base_MspDeInit

Function Name	void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Base MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.17 HAL_TIM_Base_Start

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Base generation.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

57.2.18 HAL_TIM_Base_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Base generation.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

57.2.19 HAL_TIM_Base_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Start_IT
---------------	--

(TIM_HandleTypeDef * htim)

Function Description Starts the TIM Base generation in interrupt mode.

Parameters • **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values • HAL status

57.2.20 HAL_TIM_Base_Stop_IT

Function Name **HAL_StatusTypeDef HAL_TIM_Base_Stop_IT
(TIM_HandleTypeDef * htim)**

Function Description Stops the TIM Base generation in interrupt mode.

Parameters • **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values • HAL status

57.2.21 HAL_TIM_Base_Start_DMA

Function Name **HAL_StatusTypeDef HAL_TIM_Base_Start_DMA
(TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)**

Function Description Starts the TIM Base generation in DMA mode.

Parameters • **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
• **pData:** The source Buffer address.
• **Length:** The length of data to be transferred from memory to peripheral.

Return values • HAL status

57.2.22 HAL_TIM_Base_Stop_DMA

Function Name **HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA
(TIM_HandleTypeDef * htim)**

Function Description Stops the TIM Base generation in DMA mode.

Parameters • **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values • HAL status

57.2.23 HAL_TIM_OC_Init

Function Name **HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)**

Function Description Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

Parameters • **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values • HAL status

57.2.24 HAL_TIM_OC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

57.2.25 HAL_TIM_OC_MspInit

Function Name	void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.26 HAL_TIM_OC_MspDeInit

Function Name	void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.27 HAL_TIM_OC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> • 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
Return values	<ul style="list-style-type: none"> • HAL status

57.2.28 HAL_TIM_OC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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

57.2.32 HAL_TIM_OC_Stop_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the TIM Output Compare signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • 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	<code>HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • 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	<code>HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)</code>
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> • 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	<code>void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM PWM MSP.
Parameters	<ul style="list-style-type: none"> • 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	void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	Deinitializes TIM PWM MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.37 HAL_TIM_PWM_Start

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation.
Parameters	<ul style="list-style-type: none"> • 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_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

57.2.38 HAL_TIM_PWM_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • Channel: TIM Channels to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

57.2.39 HAL_TIM_PWM_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

57.2.40 HAL_TIM_PWM_Stop_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • Channel: TIM Channels to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

57.2.41 HAL_TIM_PWM_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</code>
Function Description	Starts the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • 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_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	<ul style="list-style-type: none"> • HAL status

57.2.42 HAL_TIM_PWM_Stop_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the TIM PWM signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • Channel: TIM Channels to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

57.2.43 HAL_TIM_IC_Init

Function Name	<code>HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)</code>
---------------	---

Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

57.2.44 HAL_TIM_IC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

57.2.45 HAL_TIM_IC_MspInit

Function Name	void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.46 HAL_TIM_IC_MspDeInit

Function Name	void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.47 HAL_TIM_IC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> • 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_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

57.2.48 HAL_TIM_IC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. Channel: TIM Channels to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> HAL status

57.2.49 HAL_TIM_IC_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> 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_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> HAL status

57.2.50 HAL_TIM_IC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. Channel: TIM Channels to be disabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> HAL status

57.2.51 HAL_TIM_IC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM Input Capture measurement on in DMA mode.
Parameters	<ul style="list-style-type: none"> 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_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- HAL status

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 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

57.2.53 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_REPEATITIVE: 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	void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.56 HAL_TIM_OnePulse_MspDeInit

Function Name	void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.57 HAL_TIM_OnePulse_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

57.2.58 HAL_TIM_OnePulse_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Stops the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • OutputChannel: : 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
Return values	<ul style="list-style-type: none"> • HAL status

57.2.59 HAL_TIM_OnePulse_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • 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

57.2.60 HAL_TIM_OnePulse_Stop_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)</code>
Function Description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • 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	<code>HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)</code>
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> • 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	<code>HAL_StatusTypeDef HAL_TIM_Encoder_DelInit (TIM_HandleTypeDef * htim)</code>
Function Description	Deinitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none"> • 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	<code>void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. Channel: TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> • 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"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. sConfig: TIM PWM configuration structure Channel: TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> HAL status

57.2.75 HAL_TIM_OnePulse_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)
Function Description	Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. sConfig: TIM One Pulse configuration structure OutputChannel: TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected InputChannel: TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> HAL status

57.2.76 HAL_TIM_DMABurst_WriteStart

Function Name	HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)
Function Description	Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. BurstBaseAddress: 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_CNTTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_DMABASE_RCRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_BDRTIM_DMABASE_DCR

- **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 `HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop
(TIM_HandleTypeDef *htim, uint32_t BurstRequestSrc)`

Function Description Stops the TIM DMA Burst mode.

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

57.2.78 HAL_TIM_DMABurst_ReadStart

Function Name `HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart
(TIM_HandleTypeDef *htim, uint32_t BurstBaseAddress, uint32_t
BurstRequestSrc, uint32_t *BurstBuffer, uint32_t BurstLength)`

Function Description Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
- **BurstBaseAddress:** 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_SMCRTI
M_DMABASE_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTIM_D
MABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_CCERTI
M_DMABASE_CNTTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_
DMABASE_RCRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_
DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_BDRTIM_
DMABASE_DCR
- **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

57.2.79 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
 sourceTIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event sourceTIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event sourceTIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event sourceTIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event sourceTIM_EVENTSOURCE_COM: Timer COM event sourceTIM_EVENTSOURCE_TRIGGER: Timer Trigger Event sourceTIM_EVENTSOURCE_BREAK: Timer Break event sourceTIM_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"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. sClearInputConfig: pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral. Channel: specifies the TIM Channel. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> HAL status

57.2.82 HAL_TIM_ConfigClockSource

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)
Function Description	Configures the clock source to be used.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. sClockSourceConfig: pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.
Return values	<ul style="list-style-type: none"> HAL status

57.2.83 HAL_TIM_ConfigTI1Input

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)
Function Description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. TI1_Selection: Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values: TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 inputTIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)
Return values	<ul style="list-style-type: none"> HAL status

57.2.84 HAL_TIM_SlaveConfigSynchronization

Function Name	HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
Function Description	Configures the TIM in Slave mode.

Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. sSlaveConfig: pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	<ul style="list-style-type: none"> HAL status

57.2.85 HAL_TIM_SlaveConfigSynchronization_IT

Function Name	HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
Function Description	Configures the TIM in Slave mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> htim: TIM handle. sSlaveConfig: pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	<ul style="list-style-type: none"> HAL status

57.2.86 HAL_TIM_ReadCapturedValue

Function Name	uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Read the captured value from Capture Compare unit.
Parameters	<ul style="list-style-type: none"> 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_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> Captured value

57.2.87 HAL_TIM_PeriodElapsedCallback

Function Name	void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)
Function Description	Period elapsed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> None

57.2.88 HAL_TIM_OC_DelayElapsedCallback

Function Name	void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)
Function Description	Output Compare callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.89 HAL_TIM_IC_CaptureCallback

Function Name	void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)
Function Description	Input Capture callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.90 HAL_TIM_PWM_PulseFinishedCallback

Function Name	void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)
Function Description	PWM Pulse finished callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.91 HAL_TIM_TriggerCallback

Function Name	void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)
Function Description	Hall Trigger detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.92 HAL_TIM_ErrorCallback

Function Name	void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)
Function Description	Timer error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

57.2.93 HAL_TIM_Base_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Base state.

Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state

57.2.94 HAL_TIM_OC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM OC state.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state

57.2.95 HAL_TIM_PWM_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM PWM state.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state

57.2.96 HAL_TIM_IC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Input Capture state.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state

57.2.97 HAL_TIM_OnePulse_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM One Pulse Mode state.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state

57.2.98 HAL_TIM_Encoder_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Encoder Mode state.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that

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 TIx clock sources
`TIM_CLOCKPOLARITY_FALLING` Polarity for TIx clock sources
`TIM_CLOCKPOLARITY_BOTHEDGE` Polarity for TIx clock sources

TIM Clock Prescaler

`TIM_CLOCKPRESCALER_DIV1` No prescaler is used

TIM_CLOCKPRESCALER_DIV2	Prescaler for External ETR Clock: Capture performed once every 2 events.
TIM_CLOCKPRESCALER_DIV4	Prescaler for External ETR Clock: Capture performed once every 4 events.
TIM_CLOCKPRESCALER_DIV8	Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_ETRMODE2
TIM_CLOCKSOURCE_INTERNAL
TIM_CLOCKSOURCE_ITR0
TIM_CLOCKSOURCE_ITR1
TIM_CLOCKSOURCE_ITR2
TIM_CLOCKSOURCE_ITR3
TIM_CLOCKSOURCE_TI1ED
TIM_CLOCKSOURCE_TI1
TIM_CLOCKSOURCE_TI2
TIM_CLOCKSOURCE_ETRMODE1

TIM Commutation Source

TIM_COMMUTATION_TRGI
TIM_COMMUTATION_SOFTWARE

TIM Counter Mode

TIM_COUNTERMODE_UP
TIM_COUNTERMODE_DOWN
TIM_COUNTERMODE_CENTERALIGNED1
TIM_COUNTERMODE_CENTERALIGNED2
TIM_COUNTERMODE_CENTERALIGNED3

TIM DMA Base address

TIM_DMABASE_CR1
TIM_DMABASE_CR2
TIM_DMABASE_SMCR
TIM_DMABASE_DIER
TIM_DMABASE_SR
TIM_DMABASE_EGR
TIM_DMABASE_CCMR1
TIM_DMABASE_CCMR2
TIM_DMABASE_CCER
TIM_DMABASE_CNT

TIM_DMABASE_PSC
TIM_DMABASE_ARR
TIM_DMABASE_RCR
TIM_DMABASE_CCR1
TIM_DMABASE_CCR2
TIM_DMABASE_CCR3
TIM_DMABASE_CCR4
TIM_DMABASE_BDTR
TIM_DMABASE_DCR
TIM_DMABASE_OR

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER
TIM_DMABURSTLENGTH_2TRANSFERS
TIM_DMABURSTLENGTH_3TRANSFERS
TIM_DMABURSTLENGTH_4TRANSFERS
TIM_DMABURSTLENGTH_5TRANSFERS
TIM_DMABURSTLENGTH_6TRANSFERS
TIM_DMABURSTLENGTH_7TRANSFERS
TIM_DMABURSTLENGTH_8TRANSFERS
TIM_DMABURSTLENGTH_9TRANSFERS
TIM_DMABURSTLENGTH_10TRANSFERS
TIM_DMABURSTLENGTH_11TRANSFERS
TIM_DMABURSTLENGTH_12TRANSFERS
TIM_DMABURSTLENGTH_13TRANSFERS
TIM_DMABURSTLENGTH_14TRANSFERS
TIM_DMABURSTLENGTH_15TRANSFERS
TIM_DMABURSTLENGTH_16TRANSFERS
TIM_DMABURSTLENGTH_17TRANSFERS
TIM_DMABURSTLENGTH_18TRANSFERS

TIM DMA sources

TIM_DMA_UPDATE
TIM_DMA_CC1
TIM_DMA_CC2
TIM_DMA_CC3
TIM_DMA_CC4
TIM_DMA_COM

TIM_DMA_TRIGGER

TIM Encoder Mode

TIM_ENCODERMODE_TI1

TIM_ENCODERMODE_TI2

TIM_ENCODERMODE_TI12

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED Polarity for ETR source

TIM_ETRPOLARITY_NONINVERTED Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1 No prescaler is used

TIM_ETRPRESCALER_DIV2 ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4 ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8 ETR input source is divided by 8

TIM Event Source

TIM_EVENTSOURCE_UPDATE

TIM_EVENTSOURCE_CC1

TIM_EVENTSOURCE_CC2

TIM_EVENTSOURCE_CC3

TIM_EVENTSOURCE_CC4

TIM_EVENTSOURCE_COM

TIM_EVENTSOURCE_TRIGGER

TIM_EVENTSOURCE_BREAK

TIM_EVENTSOURCE_BREAK2

TIM Exported Macros

**_HAL_TIM_RESET_HANDLE_ST
ATE** **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_ICPRESCALERVALUE`
`TIM_SET_CAPTUREPOLARITY`
`TIM_RESET_CAPTUREPOLARITY`
`_HAL_TIM_SET_COUNTER`

Description:

- Sets the TIM Counter Register value on runtime.

Parameters:

- `_HANDLE_`: TIM handle.
- `_COUNTER_`: specifies the Counter register new value.

Return value:

- None

`_HAL_TIM_GET_COUNTER`

Description:

- Gets the TIM Counter Register value on runtime.

Parameters:

- `_HANDLE_`: TIM handle.

Return value:

- None

`_HAL_TIM_SET_AUTORELOAD`

Description:

- Sets the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:

- `_HANDLE_`: TIM handle.
- `_AUTORELOAD_`: specifies the Counter register new value.

Return value:

- None

`_HAL_TIM_GET_AUTORELOAD`

Description:

- Gets the TIM Autoreload Register value on runtime.

Parameters:

- `__HANDLE__`: TIM handle.

Return value:

- None

`__HAL_TIM_SET_CLOCKDIVISION`
N

Description:

- Sets the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- `__HANDLE__`: TIM handle.
- `__CKD__`: specifies the clock division value. This parameter can be one of the following value:
 - `TIM_CLOCKDIVISION_DIV1`
 - `TIM_CLOCKDIVISION_DIV2`
 - `TIM_CLOCKDIVISION_DIV4`

Return value:

- None

`__HAL_TIM_GET_CLOCKDIVISION`
N

Description:

- Gets the TIM Clock Division value on runtime.

Parameters:

- `__HANDLE__`: TIM handle.

Return value:

- None

`__HAL_TIM_SET_ICPRESCALER`

Description:

- Sets the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: : TIM Channels to be configured. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: TIM Channel 1 selected
 - `TIM_CHANNEL_2`: TIM Channel 2 selected
 - `TIM_CHANNEL_3`: TIM Channel 3 selected
 - `TIM_CHANNEL_4`: TIM Channel 4 selected
- `__ICPSC__`: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:

- TIM_ICPSC_DIV1: no prescaler
- TIM_ICPSC_DIV2: capture is done once every 2 events
- TIM_ICPSC_DIV4: capture is done once every 4 events
- TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

`_HAL_TIM_GET_ICPRESCALER`

Description:

- Gets the TIM Input Capture prescaler on runtime.

Parameters:

- `_HANDLE_`: TIM handle.
- `_CHANNEL_`: : TIM Channels to be configured. This parameter can be one of the following values:
 - `TIM_CHANNEL_1`: get input capture 1 prescaler value
 - `TIM_CHANNEL_2`: get input capture 2 prescaler value
 - `TIM_CHANNEL_3`: get input capture 3 prescaler value
 - `TIM_CHANNEL_4`: get input capture 4 prescaler value

Return value:

- None

`_HAL_TIM_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 TIx source
TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for TIx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for TIx source

TIM Interrupt definition

TIM_IT_UPDATE
TIM_IT_CC1
TIM_IT_CC2
TIM_IT_CC3
TIM_IT_CC4
TIM_IT_COM
TIM_IT_TRIGGER
TIM_IT_BREAK

TIM Private macros to check input parameters

IS_TIM_COUNTER_MODE
IS_TIM_CLOCKDIVISION_DIV
IS_TIM_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_REPEATITIVE

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_OCPOLARITY_HIGH
TIM_OCPOLARITY_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 TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_FALLING	Polarity for TIxFPx or TI1_ED trigger sources
TIM_TRIGGERPOLARITY_BOTHEDGE	Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1	No prescaler is used
TIM_TRIGGERPRESCALER_DIV2	Prescaler for External ETR Trigger: Capture performed once every 2 events.
TIM_TRIGGERPRESCALER_DIV4	Prescaler for External ETR Trigger: Capture performed once every 4 events.
TIM_TRIGGERPRESCALER_DIV8	Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0
TIM_TS_ITR1
TIM_TS_ITR2
TIM_TS_ITR3
TIM_TS_TI1F_ED
TIM_TS_TI1FP1
TIM_TS_TI2FP2
TIM_TS_ETRF
TIM_TS_NONE

58 HAL TIM Extension Driver

58.1 TIMEEx 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 [*TIMEEx_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 TIME_x 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_OCN_Start(),
HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OC_Start_IT()
 - Complementary PWM generation : HAL_TIMEx_PWMN_Start(),
HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
 - Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(),
HAL_TIMEx_OnePulseN_Start_IT()
 - Hall Sensor output : HAL_TIMEx_HallSensor_Start(),
HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

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_OCN_Start\(\)`](#)
- [`HAL_TIMEx_OCN_Stop\(\)`](#)
- [`HAL_TIMEx_OCN_Start_IT\(\)`](#)
- [`HAL_TIMEx_OCN_Stop_IT\(\)`](#)
- [`HAL_TIMEx_OCN_Start_DMA\(\)`](#)
- [`HAL_TIMEx_OCN_Stop_DMA\(\)`](#)

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_ConfigCommuteEvent\(\)*](#)
- [*HAL_TIMEx_ConfigCommuteEvent_IT\(\)*](#)
- [*HAL_TIMEx_ConfigCommuteEvent_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_CommputationCallback\(\)*](#)
- [*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	<code>HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, TIM_HallSensor_InitTypeDef * sConfig)</code>
Function Description	Initializes the TIM Hall Sensor Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • sConfig: TIM Hall Sensor configuration structure
Return values	<ul style="list-style-type: none"> • HAL status

58.2.11 HAL_TIMEx_HallSensor_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)</code>
Function Description	DeInitializes the TIM Hall Sensor interface.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- HAL status

58.2.12 HAL_TIMEx_HallSensor_MspInit

Function Name	<code>void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)</code>
Function Description	Initializes the TIM Hall Sensor MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- None

58.2.13 HAL_TIMEx_HallSensor_MspDeInit

Function Name	void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Hall Sensor MSP.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

58.2.14 HAL_TIMEx_HallSensor_Start

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Hall Sensor Interface.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

58.2.15 HAL_TIMEx_HallSensor_Stop

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Hall sensor Interface.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

58.2.16 HAL_TIMEx_HallSensor_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)
Function Description	Starts the TIM Hall Sensor Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

58.2.17 HAL_TIMEx_HallSensor_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Hall Sensor Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

58.2.18 HAL_TIMEx_HallSensor_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA
---------------	---

(TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function Description	Starts the TIM Hall Sensor Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • pData: The destination Buffer address. • Length: The length of data to be transferred from TIM peripheral to memory.
Return values	<ul style="list-style-type: none"> • HAL status

58.2.19 HAL_TIMEx_HallSensor_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Hall Sensor Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL status

58.2.20 HAL_TIMEx_OCN_Start

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> • 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
Return values	<ul style="list-style-type: none"> • HAL status

58.2.21 HAL_TIMEx_OCN_Stop

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

58.2.22 HAL_TIMEx_OCN_Start_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • 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
Return values	<ul style="list-style-type: none"> • HAL status

58.2.23 HAL_TIMEx_OCN_Stop_IT

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)</code>
Function Description	Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

58.2.24 HAL_TIMEx_OCN_Start_DMA

Function Name	<code>HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)</code>
Function Description	Starts the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

58.2.25 HAL_TIMEx_OCN_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation in DMA mode on the complementary output.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

58.2.26 HAL_TIMEx_PWMN_Start

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> 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
Return values	<ul style="list-style-type: none"> HAL status

58.2.27 HAL_TIMEx_PWMN_Stop

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation on the complementary output.
Parameters	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HAL status

58.2.28 HAL_TIMEx_PWMN_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation in interrupt mode on the complementary output.
Parameters	<ul style="list-style-type: none"> 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

58.2.29 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

58.2.35 HAL_TIMEx_OnePulseN_Stop_IT



Function Name	HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.
Parameters	<ul style="list-style-type: none"> 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 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> HAL status

58.2.36 HAL_TIMEx_ConfigCommutationEvent

Function Name	HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function Description	Configure the TIM commutation event sequence.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. InputTrigger: the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selectedTIM_TS_ITR1: Internal trigger 1 selectedTIM_TS_ITR2: Internal trigger 2 selectedTIM_TS_ITR3: Internal trigger 3 selectedTIM_TS_NONE: No trigger is needed CommutationSource: the Commutation Event source. This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface TimerTIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> 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.

58.2.37 HAL_TIMEx_ConfigCommutationEvent_IT

Function Name	HAL_StatusTypeDef HAL_TIMEx_ConfigCommutationEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)
Function Description	Configure the TIM commutation event sequence with interrupt.

Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. InputTrigger: the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selectedTIM_TS_ITR1: Internal trigger 1 selectedTIM_TS_ITR2: Internal trigger 2 selectedTIM_TS_ITR3: Internal trigger 3 selectedTIM_TS_NONE: No trigger is needed CommutationSource: the Commutation Event source. This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface TimerTIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> 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.

58.2.38 HAL_TIMEx_ConfigCommuteEvent_DMA

Function Name	HAL_StatusTypeDef HAL_TIMEx_ConfigCommuteEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommuteSource)
Function Description	Configure the TIM commutation event sequence with DMA.
Parameters	<ul style="list-style-type: none"> htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. InputTrigger: the Internal trigger corresponding to the Timer Interfacing with the Hall sensor. This parameter can be one of the following values: TIM_TS_ITR0: Internal trigger 0 selectedTIM_TS_ITR1: Internal trigger 1 selectedTIM_TS_ITR2: Internal trigger 2 selectedTIM_TS_ITR3: Internal trigger 3 selectedTIM_TS_NONE: No trigger is needed CommutationSource: the Commutation Event source. This parameter can be one of the following values: TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface TimerTIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer)

configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

- : The user should configure the DMA in his own software, in This function only the COMDE bit is set

58.2.39 HAL_TIM_OC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

58.2.40 HAL_TIM_PWM_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)
Function Description	Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • sConfig: TIM PWM configuration structure • Channel: TIM Channels to be enabled. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

58.2.41 HAL_TIM_ConfigOCrefClear

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)
Function Description	Configures the OCRef clear feature.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module. • sClearInputConfig: pointer to a

- TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel. This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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, OSSI/OSSR State and the AOE(automatic output enable).
Parameters	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 synchronousTIM_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 | <ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HAL status

58.2.46 HAL_TIMEx_CommmutationCallback

Function Name	void HAL_TIMEx_CommmutationCallback (TIM_HandleTypeDef * htim)
Function Description	Hall commutation changed callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • None

58.2.48 HAL_TIMEx_DMACommutationCplt

Function Name	<code>void HAL_TIMEx_DMACommutationCplt(DMA_HandleTypeDef * hdma)</code>
Function Description	TIM DMA Commutation callback.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> • None

58.2.49 HAL_TIMEx_HallSensor_GetState

Function Name	<code>HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState(TIM_HandleTypeDef * htim)</code>
Function Description	Return the TIM Hall Sensor interface state.
Parameters	<ul style="list-style-type: none"> • htim: pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.
Return values	<ul style="list-style-type: none"> • HAL state

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_DMAPause\(\)*](#)
- [*HAL_UART_DMAResume\(\)*](#)
- [*HAL_UART_DMAStop\(\)*](#)
- [*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	HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)
Function Description	Initializes the UART mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • HAL status

59.2.6 HAL_HalfDuplex_Init

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)
Function Description	Initializes the half-duplex mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle .
Parameters	<ul style="list-style-type: none"> • huart: UART handle
Return values	<ul style="list-style-type: none"> • HAL status

59.2.7 HAL_LIN_Init

Function Name	HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)
---------------	--

Function Description	Initializes the LIN mode according to the specified parameters in the <code>UART_InitTypeDef</code> and creates the associated handle .
Parameters	<ul style="list-style-type: none"> • huart: uart handle • BreakDetectLength: 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
Return values	<ul style="list-style-type: none"> • HAL status

59.2.8 HAL_MultiProcessor_Init

Function Name	<code>HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)</code>
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"> • huart: UART handle • Address: UART node address (4-, 6-, 7- or 8-bit long) • WakeUpMethod: specifies the UART wakeup method. This parameter can be one of the following values: <code>UART_WAKEUPMETHOD_IDLELINE</code>: WakeUp by an idle line detection <code>UART_WAKEUPMETHOD_ADDRESSMARK</code>: WakeUp by an address mark
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function. • If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection

59.2.9 HAL_UART_DeInit

Function Name	<code>HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)</code>
Function Description	Deinitializes the UART peripheral.
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • HAL status

59.2.10 HAL_UART_MspInit

Function Name	<code>void HAL_UART_MspInit (UART_HandleTypeDef * huart)</code>
Function Description	UART MSP Init.
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • None

59.2.11 HAL_UART_MspDeInit

Function Name	<code>void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)</code>
Function Description	UART MSP DeInit.
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • None

59.2.12 HAL_UART_Transmit

Function Name	<code>HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • huart: uart handle • pData: pointer to data buffer • Size: amount of data to be sent • Timeout: : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

59.2.13 HAL_UART_Receive

Function Name	<code>HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)</code>
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • huart: uart handle • pData: pointer to data buffer • Size: amount of data to be received • Timeout: : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

59.2.14 HAL_UART_Transmit_IT

Function Name	<code>HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</code>
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • huart: uart handle • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

59.2.15 HAL_UART_Receive_IT

Function Name	<code>HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)</code>
Function Description	Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> • huart: uart handle

- **pData:** pointer to data buffer
- **Size:** amount of data to be received
- HAL status

Return values

59.2.16 HAL_UART_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • huart: uart handle • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

59.2.17 HAL_UART_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> • huart: uart handle • pData: pointer to data buffer • Size: amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

59.2.18 HAL_UART_DMAPause

Function Name	HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • huart: UART handle
Return values	<ul style="list-style-type: none"> • None

59.2.19 HAL_UART_DMAResume

Function Name	HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • huart: UART handle
Return values	<ul style="list-style-type: none"> • None

59.2.20 HAL_UART_DMAStop

Function Name	HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)
---------------	--

Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • huart: UART handle
Return values	<ul style="list-style-type: none"> • None

59.2.21 HAL_UART_IRQHandler

Function Name	void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)
Function Description	This function handles UART interrupt request.
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • None

59.2.22 UART_WaitOnFlagUntilTimeout

Function Name	HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Timeout)
Function Description	This function handles UART Communication Timeout.
Parameters	<ul style="list-style-type: none"> • huart: UART handle • Flag: specifies the UART flag to check. • Status: The new Flag status (SET or RESET). • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

59.2.23 HAL_UART_TxCpltCallback

Function Name	void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • None

59.2.24 HAL_UART_TxHalfCpltCallback

Function Name	void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • huart: UART handle
Return values	<ul style="list-style-type: none"> • None

59.2.25 HAL_UART_RxCpltCallback

Function Name	void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> • huart: uart handle

Return values	<ul style="list-style-type: none">None
---------------	--

59.2.26 HAL_UART_RxHalfCpltCallback

Function Name	void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">huart: UART handle
Return values	<ul style="list-style-type: none">None

59.2.27 HAL_UART_ErrorCallback

Function Name	void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)
Function Description	UART error callbacks.
Parameters	<ul style="list-style-type: none">huart: uart handle
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">huart: UART handle
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">huart: uart handle
Return values	<ul style="list-style-type: none">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	<ul style="list-style-type: none">huart: uart handle
Return values	<ul style="list-style-type: none">HAL status

59.2.31 HAL_UART_GetState

Function Name	HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)
Function Description	return the UART state
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • HAL state

59.2.32 HAL_UART_GetError

Function Name	uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)
Function Description	Return the UART error code.
Parameters	<ul style="list-style-type: none"> • huart: : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	<ul style="list-style-type: none"> • UART Error Code

59.2.33 UART_SetConfig

Function Name	HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)
Function Description	Configure the UART peripheral.
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • None

59.2.34 UART_AdvFeatureConfig

Function Name	void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)
Function Description	Configure the UART peripheral advanced features.
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • None

59.2.35 UART_CheckIdleState

Function Name	HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)
Function Description	Check the UART Idle State.
Parameters	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • HAL status

59.2.36 HAL_HalfDuplex_EnableTransmitter

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)
Function Description	Enables the UART transmitter and disables the UART receiver.
Parameters	<ul style="list-style-type: none"> • huart: UART handle

Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • huart: UART handle
Return values	<ul style="list-style-type: none"> • HAL status

59.2.38 HAL_LIN_SendBreak

Function Name	HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)
Function Description	Transmits break characters.
Parameters	<ul style="list-style-type: none"> • huart: UART handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • huart: uart handle
Return values	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • huart: : pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	<ul style="list-style-type: none"> • 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_ON0X7FFFRAME
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

<code>HAL_UART_ERROR_NE</code>	Noise error
<code>HAL_UART_ERROR_FE</code>	frame error
<code>HAL_UART_ERROR_ORE</code>	Overrun error
<code>HAL_UART_ERROR_DMA</code>	DMA transfer error

UART Exported Macros

<code>__HAL_UART_RESET_HANDLE_STAT</code>	Description: • Reset UART handle state.
	Parameters: • <code>__HANDLE__</code> : UART handle.
	Return value: • None
<code>__HAL_UART_FLUSH_DRREGISTER</code>	Description: • Flush the UART Data registers.
	Parameters: • <code>__HANDLE__</code> : specifies the UART Handle.
<code>__HAL_UART_CLEAR_IT</code>	Description: • Clears the specified UART ISR flag, in setting the proper ICR register flag.
	Parameters: • <code>__HANDLE__</code> : specifies the UART Handle. • <code>__FLAG__</code> : 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: <ul style="list-style-type: none">- <code>UART_CLEAR_PEF</code>: Parity Error Clear Flag- <code>UART_CLEAR_FEF</code>: Framing Error Clear Flag- <code>UART_CLEAR_NEF</code>: Noise detected Clear Flag- <code>UART_CLEAR_OREF</code>: OverRun Error Clear Flag- <code>UART_CLEAR_IDLEF</code>: IDLE line detected Clear Flag- <code>UART_CLEAR_TCF</code>: Transmission Complete Clear Flag- <code>UART_CLEAR_LBDF</code>: LIN Break Detection Clear Flag- <code>UART_CLEAR_CTSF</code>: CTS Interrupt Clear Flag- <code>UART_CLEAR_RTOF</code>: 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

Handle.

Return value:

- None

[__HAL_UART_GET_FLAG](#)

Description:

- Checks whether the specified UART flag is set or not.

Parameters:

- [__HANDLE__](#): specifies the UART Handle.
- [__FLAG__](#): specifies the flag to check. This parameter can be one of the following values:
 - [UART_FLAG_RXACK](#): 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).

_HAL_UART_ENABLE_IT

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

_HAL_UART_DISABLE_IT

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

- interrupt
- UART_IT_TXE: Transmit Data Register empty interrupt
- UART_IT_TC: Transmission complete interrupt
- UART_IT_RXNE: Receive Data register not empty interrupt
- UART_IT_IDLE: Idle line detection interrupt
- UART_IT_PE: Parity Error interrupt
- UART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

[__HAL_UART_GET_IT](#)**Description:**

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

Parameters:

- [__HANDLE__](#): specifies the UART Handle.
- [__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_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:

- None

`_HAL_UART_ONE_BIT_SAMPLE_ENA
BLE`

Description:

- Enables the UART one bit sample method.

Parameters:

- `_HANDLE_`: specifies the UART Handle.

Return value:

- None

`_HAL_UART_ONE_BIT_SAMPLE_DIS
ABLE`

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_EN
ABLE`

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 Interruptions Flag Mask

UART_IT_MASK

UART Interrupts Definition

UART_IT_PE

UART_IT_TXE

UART_IT_TC

UART_IT_RXNE

UART_IT_IDLE

UART_IT_LBD

UART_IT_CTS

UART_IT_CM

UART_IT_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

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

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_RXINV
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 UARTE_x Firmware driver defines

60.1.1 UARTE_x

UARTE_x 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.

UARTE_x WakeUp Address Length

UART_ADDRESS_DETECT_4B

UART_ADDRESS_DETECT_7B

IS_UART_ADDRESSLENGTH_DETECT

UARTE_x 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	HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * huart)
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"> • husart: USART handle
Return values	<ul style="list-style-type: none"> • HAL status

61.2.6 HAL_USART_DelInit

Function Name	HAL_StatusTypeDef HAL_USART_DelInit (USART_HandleTypeDef * huart)
Function Description	DeInitializes the USART peripheral.
Parameters	<ul style="list-style-type: none"> • huart: USART handle
Return values	<ul style="list-style-type: none"> • HAL status

61.2.7 HAL_USART_MspInit

Function Name	void HAL_USART_MspInit (USART_HandleTypeDef * huart)
Function Description	USART MSP Init.
Parameters	<ul style="list-style-type: none"> • huart: USART handle
Return values	<ul style="list-style-type: none"> • None

61.2.8 HAL_USART_MspDelInit

Function Name	void HAL_USART_MspDelInit (USART_HandleTypeDef * huart)
Function Description	USART MSP DelInit.
Parameters	<ul style="list-style-type: none"> • huart: USART handle
Return values	<ul style="list-style-type: none"> • None

61.2.9 HAL_USART_CheckIdleState

Function Name	HAL_StatusTypeDef HAL_USART_CheckIdleState (USART_HandleTypeDef * huart)
Function Description	

61.2.10 HAL_USART_Transmit

Function Name	HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * huart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)
Function Description	Simplex Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • huart: USART handle • pTxData: pointer to data buffer • Size: amount of data to be sent • Timeout: : Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

61.2.11 HAL_USART_Receive

Function Name	HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * huart, 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"> husart: USART handle pRxData: pointer to data buffer Size: amount of data to be received Timeout: : Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> To receive synchronous data, dummy data are simultaneously transmitted

61.2.12 HAL_USART_TransmitReceive

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Full-Duplex Send and Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> husart: USART handle pTxData: pointer to TX data buffer pRxData: pointer to RX data buffer Size: amount of data to be sent (same amount to be received) Timeout: : Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

61.2.13 HAL_USART_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Send an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> husart: USART handle pTxData: pointer to data buffer Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status

61.2.14 HAL_USART_Receive_IT

Function Name	HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Receive an amount of data in blocking mode To receive synchronous data, dummy data are simultaneously transmitted.
Parameters	<ul style="list-style-type: none"> husart: USART handle pRxData: pointer to data buffer Size: amount of data to be received
Return values	<ul style="list-style-type: none"> HAL status

61.2.15 HAL_USART_TransmitReceive_IT

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Send and Receive an amount of data in interrupt mode.
Parameters	<ul style="list-style-type: none"> husart: USART handle pTxData: pointer to TX data buffer pRxData: pointer to RX data buffer Size: amount of data to be sent (same amount to be received)
Return values	<ul style="list-style-type: none"> HAL status

61.2.16 HAL_USART_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Send an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> husart: USART handle pTxData: pointer to data buffer Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> HAL status

61.2.17 HAL_USART_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Receive an amount of data in DMA mode.
Parameters	<ul style="list-style-type: none"> husart: USART handle pRxData: pointer to data buffer Size: amount of data to be received
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position) The USART DMA transmit stream must be configured in order to generate the clock for the slave.

61.2.18 HAL_USART_TransmitReceive_DMA

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Transmit Receive an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> husart: USART handle pTxData: pointer to TX data buffer pRxData: pointer to RX data buffer

- **Size:** amount of data to be received/sent
- Return values
 - HAL status
- Notes
 - When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

61.2.19 HAL_USART_DMAPause

Function Name	HAL_StatusTypeDef HAL_USART_DM_PAUSE (USART_HandleTypeDef * husart)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.20 HAL_USART_DMAResume

Function Name	HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.21 HAL_USART_DMAStop

Function Name	HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.22 HAL_USART_IRQHandler

Function Name	void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)
Function Description	This function handles USART interrupt request.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.23 HAL_USART_TxCpltCallback

Function Name	void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.24 HAL_USART_TxHalfCpltCallback

Function Name	void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.25 HAL_USART_RxCpltCallback

Function Name	void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.26 HAL_USART_RxHalfCpltCallback

Function Name	void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart: usart handle
Return values	<ul style="list-style-type: none">• None

61.2.27 HAL_USART_TxRxCpltCallback

Function Name	void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx/Rx Transfers completed callback for the non-blocking process.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.28 HAL_USART_ErrorCallback

Function Name	void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)
Function Description	USART error callbacks.
Parameters	<ul style="list-style-type: none">• husart: USART handle
Return values	<ul style="list-style-type: none">• None

61.2.29 HAL_USART_GetState

Function Name	HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)
Function Description	return the USART state

Parameters	<ul style="list-style-type: none"> • husart: USART handle
Return values	<ul style="list-style-type: none"> • HAL state

61.2.30 HAL_USART_GetError

Function Name	<code>uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)</code>
Function Description	Return the USART error code.
Parameters	<ul style="list-style-type: none"> • husart: : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.
Return values	<ul style="list-style-type: none"> • USART Error Code

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

`_HAL_USART_RESET_HANDLE_ST
ATE` **Description:**

- Reset USART handle state.

Parameters:

- `_HANDLE_`: USART handle.

Return value:

- None

[__HAL_USART_GET_FLAG](#)**Description:**

- Checks whether the specified USART flag is set or not.

Parameters:

- [__HANDLE__](#): specifies the USART Handle
- [__FLAG__](#): specifies the flag to check. This parameter can be one of the following values:
 - [USART_FLAG_RXACK](#): 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)

Return value:

- None

[__HAL_USART_DISABLE_IT](#)

- 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](#)

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

Parameters:

- __HANDLE__: specifies the USART Handle
- __IT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
 - USART_IT_TXE: Transmit Data Register empty interrupt
 - USART_IT_TC: Transmission complete interrupt
 - USART_IT_RXNE: Receive Data register not empty interrupt
 - USART_IT_IDLE: Idle line detection interrupt
 - USART_IT_ORE: OverRun Error interrupt
 - USART_IT_NE: Noise Error interrupt
 - USART_IT_FE: Framing Error interrupt
 - USART_IT_PE: Parity Error interrupt

Return value:

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

[__HAL_USART_GET_IT_SOURCE](#)**Description:**

- Checks whether the specified USART interrupt source is enabled.
- Parameters:**
- [__HANDLE__](#): specifies the USART Handle.
 - [__IT__](#): specifies the USART interrupt source to check. This parameter can be one of the following values:
 - [USART_IT_TXE](#): Transmit Data Register empty interrupt
 - [USART_IT_TC](#): Transmission complete interrupt
 - [USART_IT_RXNE](#): Receive Data register not empty interrupt
 - [USART_IT_IDLE](#): Idle line detection interrupt
 - [USART_IT_ORE](#): OverRun Error interrupt
 - [USART_IT_NE](#): Noise Error interrupt
 - [USART_IT_FE](#): Framing Error interrupt
 - [USART_IT_PE](#): Parity Error interrupt

Return value:

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

[__HAL_USART_CLEAR_IT](#)**Description:**

- Clears the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- [__HANDLE__](#): specifies the USART Handle.
- [__IT_CLEAR__](#): specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
 - [USART_CLEAR_PEF](#): Parity Error Clear Flag
 - [USART_CLEAR_FEF](#): Framing Error Clear Flag
 - [USART_CLEAR_NEF](#): Noise detected Clear Flag
 - [USART_CLEAR_OREF](#): OverRun Error Clear Flag
 - [USART_CLEAR_IDLEF](#): IDLE line detected Clear Flag
 - [USART_CLEAR_TCF](#): Transmission Complete Clear Flag
 - [USART_CLEAR_CTSF](#): CTS Interrupt Clear Flag

Return value:

- None

__HAL_USART_SEND_REQ**Description:**

- Set a specific USART request flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
 - USART_RXDATA_FLUSH_REQUEST : Receive Data flush Request
 - USART_TXDATA_FLUSH_REQUEST: Transmit data flush Request

Return value:

- None

__HAL_USART_ENABLE**Description:**

- Enable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_DISABLE**Description:**

- Disable USART.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

USART Flags

USART_FLAG_RXACK

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_RXACK_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 USARTEx Firmware driver defines

62.1.1 USARTEx

USARTEx 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`

USARTEx 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_HandleTypeDefDef

Data Fields

- *WWDG_TypeDef * Instance*
- *WWDG_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_WWDG_StateTypeDef State*

Field Documentation

- ***WWDG_TypeDef* WWDG_HandleTypeDefDef::Instance***
Register base address
- ***WWDG_InitTypeDef WWDG_HandleTypeDefDef::Init***
WWDG required parameters
- ***HAL_LockTypeDef WWDG_HandleTypeDefDef::Lock***
WWDG locking object
- ***__IO HAL_WWDG_StateTypeDef WWDG_HandleTypeDefDef::State***
WWDG communication state

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.
- WWDRST flag in RCC_CSR register can be used to inform when a WWdg reset occurs.
- The WWdg counter input clock is derived from the APB clock divided by a programmable prescaler.
- WWdg clock (Hz) = PCLK1 / (4096 * Prescaler)
- WWdg timeout (mS) = 1000 * Counter / WWdg clock
- WWdg Counter refresh is allowed between the following limits :
 - min time (mS) = 1000 * (Counter - Window) / WWdg clock
 - max time (mS) = 1000 * (Counter - 0x40) / WWdg clock
- Min-max timeout value at 50 MHz(PCLK1): 81.9 us / 41.9 ms

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	HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwdg)
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"> • hwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

63.2.7 HAL_WWDG_DeInit

Function Name	HAL_StatusTypeDef HAL_WWDG_DeInit (WWDG_HandleTypeDef * hwdg)
Function Description	DeInitializes the WWDG peripheral.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified

WWDG module.

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

63.2.8 HAL_WWDG_MspInit

Function Name	void HAL_WWDG_MspInit (WWDG_HandleTypeDefDef * hwdg)
Function Description	Initializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWDG_HandleTypeDefDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • None

63.2.9 HAL_WWDG_MspDeInit

Function Name	void HAL_WWDG_MspDeInit (WWDG_HandleTypeDefDef * hwdg)
Function Description	Deinitializes the WWDG MSP.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWDG_HandleTypeDefDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • None

63.2.10 HAL_WWDG_WakeupCallback

Function Name	void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDefDef * hwdg)
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWDG_HandleTypeDefDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • None

63.2.11 HAL_WWDG_Start

Function Name	HAL_StatusTypeDef HAL_WWDG_Start (WWDG_HandleTypeDefDef * hwdg)
Function Description	Starts the WWDG.
Parameters	<ul style="list-style-type: none"> • hwdg: pointer to a WWDG_HandleTypeDefDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

63.2.12 HAL_WWDG_Start_IT

Function Name	HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDefDef * hwdg)
Function Description	Starts the WWDG with interrupt enabled.

Parameters	<ul style="list-style-type: none"> hwdwg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> HAL status

63.2.13 HAL_WWDG_Refresh

Function Name	HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwdwg, uint32_t Counter)
Function Description	Refreshes the WWDG.
Parameters	<ul style="list-style-type: none"> hwdwg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module. Counter: value of counter to put in WWDG counter
Return values	<ul style="list-style-type: none"> HAL status

63.2.14 HAL_WWDG_IRQHandler

Function Name	void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwdwg)
Function Description	Handles WWDG interrupt request.
Parameters	<ul style="list-style-type: none"> hwdwg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled using __HAL_WWDG_ENABLE_IT() macro. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

63.2.15 HAL_WWDG_WakeupCallback

Function Name	void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwdwg)
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> hwdwg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> None

63.2.16 HAL_WWDG_GetState

Function Name	HAL_WWDG_StateTypeDef HAL_WWDG_GetState (WWDG_HandleTypeDef * hwdwg)
---------------	---

Function Description	Returns the WWDG state.
Parameters	<ul style="list-style-type: none"> hwwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> HAL state

63.3 WWDG Firmware driver defines

63.3.1 WWDG

WWDG Exported Macros

<code>_HAL_WWDG_RESET_HANDLE_STATE</code>	<p>Description:</p> <ul style="list-style-type: none"> Reset WWDG handle state. <p>Parameters:</p> <ul style="list-style-type: none"> <code>_HANDLE_</code>: WWDG handle <p>Return value:</p> <ul style="list-style-type: none"> None
<code>_HAL_WWDG_ENABLE</code>	<p>Description:</p> <ul style="list-style-type: none"> Enables the WWDG peripheral. <p>Parameters:</p> <ul style="list-style-type: none"> <code>_HANDLE_</code>: WWDG handle <p>Return value:</p> <ul style="list-style-type: none"> None
<code>_HAL_WWDG_DISABLE</code>	<p>Description:</p> <ul style="list-style-type: none"> Disables the WWDG peripheral. <p>Parameters:</p> <ul style="list-style-type: none"> <code>_HANDLE_</code>: WWDG handle <p>Return value:</p> <ul style="list-style-type: none"> None <p>Notes:</p> <ul style="list-style-type: none"> WARNING: This is a dummy macro for HAL code alignment. Once enable, WWDG Peripheral cannot be disabled except by a system reset.
<code>_HAL_WWDG_GET_IT</code>	<p>Description:</p> <ul style="list-style-type: none"> Gets the selected WWDG's it status. <p>Parameters:</p> <ul style="list-style-type: none"> <code>_HANDLE_</code>: WWDG handle <code>_INTERRUPT_</code>: specifies the it to check. This parameter can be one of the following values:

- WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The new state of WWDG_FLAG (SET or RESET).

_HAL_WWDG_CLEAR_IT**Description:**

- Clear the WWDG's interrupt pending bits to clear the selected interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - WWDG_FLAG_EWIF: Early wakeup interrupt flag

_HAL_WWDG_ENABLE_IT**Description:**

- Enables the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
 - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

_HAL_WWDG_DISABLE_IT**Description:**

- Disables the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to disable. This parameter can be one of the following values:
 - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment. Once enabled this interrupt cannot be disabled except by a system reset.

`_HAL_WWDG_GET_FLAG`**Description:**

- Gets the selected WWDG's flag status.

Parameters:

- `_HANDLE_`: WWDG handle
- `_FLAG_`: specifies the flag to check. This parameter can be one of the following values:
 - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

Return value:

- The new state of `WWDG_FLAG` (SET or RESET).

`_HAL_WWDG_CLEAR_FLAG`**Description:**

- Clears the WWDG's pending flags.

Parameters:

- `_HANDLE_`: WWDG handle
- `_FLAG_`: specifies the flag to clear. This parameter can be one of the following values:
 - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

Return value:

- None

`_HAL_WWDG_GET_IT_SOURCE`**Description:**

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

Parameters:

- `_HANDLE_`: WWDG Handle.
- `_INTERRUPT_`: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
 - `WWDG_IT_EWI`: Early Wakeup Interrupt

Return value:

- `state`: of `_INTERRUPT_` (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1 WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2 WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4 WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8 WWDG counter clock = (PCLK1/4096)/8

WWDG Private Macros

IS_WWDG_PRESCALER

IS_WWDG_WINDOW

IS_WWDG_COUNTER

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 require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, ...). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which 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 other series and lines, the API is split into the generic and the extension APIs. For more details, please refer to [Section 4.4: "Devices supported by HAL drivers"](#).

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: `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 function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in xx_hal_msp.c. A template is provided in the HAL driver folders (xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute __weak)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,...), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The sysTick configuration shall be adjusted by calling **HAL_RCC_ClockConfig()** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling **HAL_IncTick()** function in Systick ISR and retrieve the value of this variable by calling **HAL_GetTick()** function.

The call **HAL_GetTick()** function is mandatory when using HAL drivers with Polling Process or when using **HAL_Delay()**.

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using **HAL_Delay()** since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if **HAL_Delay()** is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use **HAL_NVIC_SetPriority()** function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call **HAL_Init()** function to initialize the system (data cache, NVIC priority,...).
2. Initialize the system clock by calling **HAL_RCC_OscConfig()** followed by **HAL_RCC_ClockConfig()**.
3. Add **HAL_IncTick()** function under **SysTick_Handler()** ISR function to enable polling process when using **HAL_Delay()** function
4. Start initializing your peripheral by calling **HAL_PPP_Init()**.
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling **HAL_PPP_MspInit()** in **xx_hal_msp.c**
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under **PPP_IRQHandler()** function in **xx_it.c**. In this case, the end-user has to implement only the callbacks functions (prefixed by **__weak**) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in **PPP_IRQHandler()** without calling **HAL_PPP_IRQHandler()**.

Can I use directly the macros defined in xx_hal_ppp.h ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypeDef structure peripheral handler be declared?

PPP_HandleTypeDef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

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